STATUS OF THESIS

Title of thesis | DESIGN AND EVALUATION OF RESOURCE ALLOCATION AND JOB SCHEDULING ALGORITHMS ON COMPUTATIONAL GRIDS

I, SYED NASIR MEHMOOD SHAH

hereby allow my thesis to be placed at the Information Resource Center (IRC) of Universiti Teknologi PETRONAS (UTP) with the following conditions:

1. The thesis becomes the property of UTP
2. The IRC of UTP may make copies of the thesis for academic purposes only.
3. This thesis is classified as

☐ Confidential

☑ Non-confidential

If this thesis is confidential, please state the reason:

_____
_____
_____

The contents of the thesis will remain confidential for _____ years.

Remarks on disclosure:

_____
_____
_____

Endorsed by

_____	_____
Signature of Author	Signature of Supervisor
Permanent Address:	Name of Supervisor:
Gillani House, E-139, Kahuta, Rawalpindi, Punjab (Pakistan)	Assoc. Prof. Dr. Ahmad Kamil B. Mahmood
Date: _____	Date: _____

UNIVERSITI TEKNOLOGI PETRONAS

DESIGN AND EVALUATION OF RESOURCE ALLOCATION AND JOB SCHEDULING ALGORITHMS ON COMPUTATIONAL GRIDS

by

SYED NASIR MEHMOOD SHAH

The undersigned certify that they have read, and recommend to the Postgraduate Studies Programme for acceptance this thesis for the fullfilment of the requirements for the degree stated.

Signature: _____

Main Supervisor: Assoc. Prof. Dr. Ahmad Kamil B. Mahmood

Signature: _____

Cosupervisor: Prof. Dr. Alan Oxley

Signature: _____

Head of Department: Dr. Mohd Fadzil Bin Hassan

Date: _____

DESIGN AND EVALUATION OF RESOURCE ALLOCATION AND JOB SCHEDULING ALGORITHMS ON COMPUTATIONAL GRIDS

by

SYED NASIR MEHMOOD SHAH

A Thesis

Submitted to the Postgraduate Studies Programme

As a requirement for the degree of

DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER AND INFORMATION SCIENCES

UNIVERSITI TEKNOLOGI PETRONAS

BANDAR SERI ISKANDAR

PERAK

JANUARY 2012

DECLARATION OF THESIS

Title of thesis | DESIGN AND EVALUATION OF RESOURCE ALLOCATION AND JOB SCHEDULING ALGORITHMS ON COMPUTATIONAL GRIDS

I, SYED NASIR MEHMOOD SHAH

hereby declare that the thesis is based on my original work except for quotations and citations, which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at UTP or other institutions.

Witnessed by

_____	_____
Signature of Author	Signature of Supervisor
Permanent Address:	Name of Supervisor:
Gillani House, E-139, Kahuta, Rawalpindi, Punjab (Pakistan)	Assoc. Prof. Dr. Ahmad Kamil B. Mahmood
Date: _____	Date: _____

To My Beloved Holy Prophet Muhammad (Peace Be Upon Him),

His Family and His Companions

My late father, May the Mercy of Allah be upon him,

My beloved mother, my sisters and my brothers

ACKNOWLEDGEMENTS

First of all, I owe my deepest gratitude to almighty ALLAH, the Most Merciful and Compassionate, the Most Gracious and Beneficent, for blessing me strength, health and will power to prevail and finish this work. May Allah accept this work, count it as a good deed and make it useful.

I would like to express my profound thanks and gratitude to my supervisors Assoc. Prof. Dr. Ahmad Kamil Bin Mahmood and Prof. Dr. Alan Oxley, without whose guidance, encouragement and affection, the thesis could not be completed. It was Dr. Ahmad Kamil, who introduced me in the area of Grid Computing, and continued helping me in every stage of my endeavours to delve into its depth.

Furthermore, a very special note of thanks and appreciation goes to my lovely mother, my sisters, my brothers and other family members, for their constant help, encouragement and love throughout my academic career. Words seem to be insufficient to thank them, for enduring all my problems with great patience and love. I can say that, my academic access and progress, and any minor or major achievement in my life is due to these people, whose endless prayers are a source of determination for me.

I am grateful to all of faculty members, staff members and my colleagues in the Computer and Information Science Department. But my special thanks go to Syed Zain, S. J. Hosseini, Hanieh Kazemi, M. Asif, Rafi Raza, Rana Shahid, Raja Shahzad, Aamir Amanat, Hafiz Tahir Ahmed, Dileep Kumar, Sami Ullah, Sajjad Ahmad, Dr. Mona, Dr. Mohamed Abdee, Dr. Yasir, Dr. Esa, Babar Nazir, Jahangir Khan, Ahmad Haruna, Helmi, Izzudin, Nurul Natrah and Okta. I owe to the members of the High Performance Computing research group for the long discussions that had provided me with enlightening views of the research problems that I was struggling with.

I am also privileged to have the opportunity to collaborate with some of the most brilliant friends Dr. M. Yonus Javed, Dr. Amir Shafi, Dr. A. J. Pal, Dani Adhipta, Ms. Nazleeni Haroon and Thayalan Sandran.

Last but not the least, I would also like to thank to all of my Pakistani and Malaysian friends, and other international students at the University Teknologi PETRONAS for their endless support and spirit of brotherhood. I shall always cherish my association and affinities with all of them and treasure the good days and happy moments spent with them.

ABSTRACT

Grid, an infrastructure for resource sharing, currently has shown its importance in many scientific applications requiring tremendously high computational power. Grid computing enables sharing, selection and aggregation of resources for solving complex and large-scale scientific problems. Grids computing, whose resources are distributed, heterogeneous and dynamic in nature, introduces a number of fascinating issues in resource management. Grid scheduling is the key issue in grid environment in which its system must meet the functional requirements of heterogeneous domains, which are sometimes conflicting in nature also, like user, application, and network. Moreover, the system must satisfy non-functional requirements like reliability, efficiency, performance, effective resource utilization, and scalability. Thus, overall aim of this research is to introduce new grid scheduling algorithms for resource allocation as well as for job scheduling for enabling a highly efficient and effective utilization of the resources in executing various applications.

The four prime aspects of this work are: firstly, a model of the grid scheduling problem for dynamic grid computing environment; secondly, development of a new web based simulator (SyedWSim), enabling the grid users to conduct a statistical analysis of grid workload traces and provides a realistic basis for experimentation in resource allocation and job scheduling algorithms on a grid; thirdly, proposal of a new grid resource allocation method of optimal computational cost using synthetic and real workload traces with respect to other allocation methods; and finally, proposal of some new job scheduling algorithms of optimal performance considering parameters like waiting time, turnaround time, response time, bounded slowdown, completion time and stretch time. The issue is not only to develop new algorithms, but also to evaluate them on an experimental computational grid, using synthetic and real workload traces, along with the other existing job scheduling algorithms. Experimental evaluation confirmed that the proposed grid scheduling algorithms possess a high degree of optimality in performance, efficiency and scalability.

ABSTRAK

Grid, merupakan sebuah infrastruktur untuk perkongsian sumber, telah menunjukkan kepentingannya pada masa kini dalam pelbagai aplikasi saintifik yang memerlukan kuasa pengkomputeran tinggi dengan pesatnya. Pengkomputeran grid membolehkan perkongsian, pemilihan dan pengumpulan sumber untuk menyelesaikan masalah saintifik yang kompleks dan berskala besar. Sumber-sumber pengkomputeran grid yang kebiasaannya teragih, pelbagai jenis dan dinamik secara semula jadi, memperkenalkan beberapa isu yang cukup menarik dalam pengurusan sumber. Penjadualan grid adalah isu utama dalam persekitaran grid di mana sesebuah sistem perlulah memenuhi keperluan fungsi kepelbagaian domain, yang kadangkala bertentangan dalam alam semula jadi seperti pengguna, aplikasi, dan rangkaian. Tambahan pula, sistem mestilah memenuhi keperluan bukan-fungsi seperti kebolehpercayaan, kecekapan, prestasi, keberkesanan dalam penggunaan sumber, dan kebolehskalaan. Oleh itu, matlamat keseluruhan kajian ini adalah untuk memperkenalkan algoritma penjadualan grid yang baru untuk peruntukan sumber seperti mana juga penjadualan tugasan yang akan membolehkan penggunaan pelbagai sumber yang sangat cekap dan berkesan dalam melaksanakan pelbagai aplikasi.

Empat aspek utama yang terkandung dalam kajian ini ialah: pertamanya, sebuah model dalam masalah penjadualan grid untuk persekitaran grid yang dinamik; kedua, pembentukan alat simulasi berasaskan web yang baru (SyedWSim), membolehkan para pengguna grid untuk menjalankan analisis statistik dalam mengesan beban kerja grid dan menyediakan asas yang realistik untuk uji kaji dalam peruntukan sumber dan algoritma penjadualan kerja di dalam sesebuah grid; ketiga, cadangan kaedah peruntukan sumber grid yang baru dalam mengoptimakan kos pengiraan yang menggunakan pengesanan beban kerja sintetik dan asli dengan bersandarkan kaedah-kaedah peruntukan yang sedia ada, dan yang terakhir, cadangan sesetengah algoritma penjadualan kerja yang baru memiliki prestasi optimum dengan mengambil kira parameter seperti masa menunggu, masa yang diambil, masa tindak balas, masa aliran, kelembapan terbatas dan masa regangan. Hal ini bukan sahaja untuk

membangunkan algoritma baru, tetapi juga untuk menilai mereka dalam persekitaran grid uji kaji, menggunakan pengesanan beban kerja sintetik dan asli bersama-sama dengan algoritma penjadualan kerja yang sedia ada. Penilaian eksperimen telah mengesahkan bahawa algoritma penjadualan yang dicadangkan memiliki pencapaian yang optimum dalam prestasi, kecekapan dan kebolehskalaan.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Chapter Overview

A grid is a computational system consisting of large number of geographically distributed and heterogeneous resources that provides dependable, pervasive, consistent, and inexpensive access to high-end computational powers, beyond the capacity of even the largest parallel computer system. Resource allocation and job scheduling are the key components of grid, which play an important role in the efficient and effective execution of various kinds of scientific and engineering applications.

This chapter presents the motivations and general context of thesis. First, it gives chapter overview, and then section 1.2 presents the background of grid scheduling. Section 1.3 highlights the motivation to do research in this area. The problem statement is furthermore detailed in section 1.4. The research questions, essentially to be answered, to solve the stated problem are discussed in section 1.6. Section 1.7 of this chapter then deals with the list of the research objectives necessary to design new scheduling algorithms as well as workflow of those activities. In Section 1.8, a description of the methodology and activities carried out to conduct the research are presented. Section 1.8 highlights the research scope of this thesis. A road map of thesis is presented in section 1.9. This chapter then is concluded in section 1.10.

1.2 Background

Computational approaches [1] have been widely used to solve several complex problems in different fields, including high-energy physics, earth system sciences,

bioinformatics, biomedical science, geosciences, astronomy and financial modeling. Computers meanwhile are used for modeling and simulation purposes in complex scientific, engineering and commerce problems, e.g., diagnosing medical conditions, controlling industrial equipment, forecasting the weather, managing stock portfolios etc.

A number of powerful parallel systems have been built with increasing number of processors and multi-core solutions in order to meet the computational demands of various scientific applications. The Top500 maintains the world's top computing machines [2]. To date, the K Computer with 548352 cores is one of the best super computers in world [3]. In spite of the huge amount of parallelism, currently available in a single institution, current applications like biomedical applications still need more computational power for execution. This then has motivated the development of a new computing paradigm called grid Computing.

Today, grids have been utilized in various scientific applications in which a large amount of data needs to be shared, managed and processed. Efficient management of distributed resources such as data, scientific instruments and devices for computation is vital for supporting complex scientific experiments while running the application in grid environments [4].

Malaysia currently has been developing the DBRAIN system for diagnosis, therapeutics and treatment of dementia-affected people with the support of grid-driven bio-computing platforms. DBRAIN is part of the national road map for knowledge grid [5]. Similarly, the Mayo Clinic is also developing a system for linking its own medical database with large number of external public and private data sources in order to provide more-effective treatments for the patients [6]. Other important scientific applications also include brain activity analysis [7], particle physics [8], aerospace design optimization [9], earth simulation etc. For these applications, grids have been used to provide feasibility to solve much larger-scale problem that usually cannot be solved on a single computer or site.

Moreover, multimedia companies can also take advantage of grid for development of animation software, especially for purposes related to visualization and rendering.

Another application is in the area of public safety, where surveillance and tracking analysis as well as the collection of data can easily be carried out using a computational grid [10].

A multitude of jobs for generating and processing large datasets is commonly present in scientific and engineering applications. Assigning specific jobs to the appropriate resources in order to achieve effective and efficient execution can be performed by the computational grid [11], [12].

Additionally, utilization of the immense computing power available in the grid can enable the nations to make the command and control centers set up for stronger and the more effective monitoring of natural disasters. At this point, it can be utilized in climatic studies through simulation in order to get early detection by studying the natural phenomena [10].

1.3 Motivation

A grid is an infrastructure for resource sharing. At present, many scientific applications require high performance in processing, which can only be achieved by using the computational grid.

For the selection and allocation of grid resources to current and future applications, resource management and job scheduling components are playing a very vital role for computational grids. They constitute the building blocks for making grids available to the society. The efficient and effective scheduling policies, when assigning different jobs to specific resources, are very important for a grid to process high computing intensive applications [13], [14].

A number of interesting challenges have been introduced to scheduling by grid computing in which the scheduling policies not only can manage the various resources needed in computing, but also can make the decisions regarding the dynamic execution of jobs. Optimization of the grid performance is dependent on the scheduling policies [11], [13], [15], [16], [17]. In order to obtain a grid environment, which works with high performance, the effective and efficient resource management

is a must. Due to the high dynamicity, scalability and heterogeneity of a grid, some challenges then arise in the development of algorithms for scheduling to be used along with grid computing [18].

In literature most of the grid scheduling algorithms proposed, are static in nature and work in centralized manner. However, these algorithms fail to work efficiently in heterogeneous, dynamic and fully distributed grid environment where the requirements of jobs and resources are difficult to predict [18], [19], [20].

To achieve high performance, there is a need to understand the factors that can affect the performance of application due to scheduling. This research work gets the motivation from the above mentioned factors which in turn prompted to design efficient and effective scheduling algorithms for dynamic grid computing environment.

1.4 Research Problem

Grid computing is the enabling technology for high performance in scientific and large-scale computing applications and introduces a number of fascinating issues to scheduling. Grid scheduling in turn acts as a vital component of a grid infrastructure. Grid scheduling plays a critical role in the efficient and effective management of resources to achieve high performance on computational grid [1], [21], [22], [23],[24], [25].

The performance of a grid can significantly be impacted by proper grid scheduling. Scheduling is difficult and challenging in grid computing due to distributed, dynamic, heterogeneous and unpredictable nature of grid resources. The scheduling problem can be viewed as a multivariate optimization problem, where the set of tasks is being assigned to a set of resources to optimize the overall execution time. Grid scheduling problem is NP-Complete in nature [26]. This scheduling problem is extensively studied and various heuristics have been proposed in the literature. Until recently, there no algorithms have been found with the ability to

provide an optimal solution for each instance in such problems within reasonable time [27], [28], [29].

Several challenges in grid scheduling make the implementation of practical systems to be quite difficult. A grid scheduling system must meet the functional requirements of heterogeneous domains (e.g., user domains, application domains, and network domains) in which these requirements sometimes become incompatible with one another. Moreover, a grid scheduling system must also satisfy non-functional requirements, such as reliability, efficiency (in terms of time consumption), performance, effectiveness in resource utilization, and scalability [13], [27], [28], [29]. Till now, no such algorithm exists, which could meet the above mentioned aspects of scheduling. Thus, it is essential to introduce new grid scheduling algorithms for effective utilization of resources and efficient execution of applications.

Four aspects of the research problem are stated below:

Firstly, propose a grid scheduling model, which can reflect every aspect of scheduling algorithm. Secondly, design of a new web based simulator to perform the statistical analysis of different grid workload traces, and can provide the realistic basis for evaluation of resource allocation and job scheduling algorithms as per the proposed grid scheduling model. Thirdly, design a new grid resource allocation method and compare its performance with some remarkable algorithms on different workload traces as per the proposed grid scheduling model. Finally, design new job scheduling algorithms and compare their performances with some remarkable algorithms on different workload traces as per the proposed grid scheduling model.

1.5 Research Questions

The performance of a grid scheduler is found to be strongly dependent on scheduling policies and the characteristics of jobs such as number of tasks, priority of tasks or the run-time of tasks [25]. In this thesis, an attempt to provide the answers to the following questions has been made:

1. What model to consider for designing and evaluating new grid resource allocation and job scheduling algorithms?

To design new grid scheduling algorithms (resource allocation and job scheduling), a suitable scheduling model is required. There is need to design a new dynamic grid scheduling model based on the standard procedures [30], [31], [32], [33]. The proposed grid scheduling model will provide the basis for designing and development of resource allocation and job scheduling algorithms; and will also facilitate this research in evaluating the performance of grid scheduling algorithms on the standard benchmark practices.

2. How can an efficient and effective utilization of grid resources be made while minimizing the cost of computation?

Grid resource allocation is an NP complete problem [26]. There is a need to design and develop an optimized resource allocation method, which will ensure an efficient allocation of resources and lower the computational cost in terms of time.

3. How can an efficient and fair job scheduler be made to maximize the performance and efficiency of a grid?

There is a need to design and develop an efficient and fair job scheduler, which would be responsible for the fair distribution of resources among user jobs, and ensure the efficient execution of those user jobs.

4. How can the proposed grid resource allocation method and job scheduling algorithms be evaluated?

A strategy of performance evaluation for testing and validating the proposed resource allocation and job scheduling algorithms on a grid under different workload traces is needed and should be designed using the standard benchmark procedures as detailed in [31], [32] , [33]. This question can be realized in three more dimensions as detailed below:

(a) How can the statistical analysis of grid workload traces be performed?

Realistic workloads traces are required to measure the efficiency, performance and scalability of scheduling algorithms. The result of a performance evaluation depends strongly on the workload used [24], [32], [33], [34]. The use of a workload that incorrectly represents the real situation may result in inaccurate performance measurements. Therefore, it is necessary to analyze the specific characteristic of the jobs (statistical analysis) that may have a strong impact on the grid scheduling policies [33]. Development of a web-based simulator is required to make this study accurate and user friendly.

(b) How can grid scheduling policies be evaluated on different architectures?

It is desirable to do a performance analysis to the extent that the results are applicable to different grid architectures. Since there is no de-facto standard for grid architectures, the results of a performance analysis on one system do not necessarily hold for other systems, especially since grid architecture is complex and large with several hardware and software layers [33].

(c) What is the performance of grid scheduling policies under different grid workload traces?

Different grid scheduling algorithms must be compared under synthetic and real workload traces [33], [34] of various sizes to draw the conclusion.

1.6 Research Objectives

To answer the research questions stated in the previous section, a number of technical objectives need to be achieved as a part of this research work, as presented below:

1. To propose a dynamic grid scheduling model to provide the basis for design and evaluation of resource allocation and job scheduling algorithms.

2. To propose new grid resource allocation method to optimize the utilization of resources and lower the cost of computation.
3. To propose new grid job scheduling algorithms that will possess a high degree of performance, efficiency and scalability.
4. To design and develop a web-based simulator to study the nature of the various grid workload traces.
5. To evaluate the proposed grid resource allocation method for different grid computing scenarios using simulation.
6. To evaluate the proposed job scheduling algorithms on an experimental grid using synthetic and real workload traces.

1.7 Research Methodology and Activities

The research objectives are realizable by considering the following research methodology and activities:

1. Conducting an extensive literature survey of work in specific areas of distributed computing, grid computing, operating system and operation research:
    a. Grid resource management
    b. Grid-wide resource sharing
    c. Transportation methods
    d. Static and dynamic workload distribution
    e. Adaptive load-balancing
    f. Divisible load scheduling
    g. Flexible (dynamic) partitions of nodes for users
    h. Static and dynamic job scheduling
2. Constructing the grid scheduling model focusing on the issues of grid scheduling.
3. Designing and developing a web-based simulator for statistical analysis of grid workload traces.
4. Formulating the mathematical model for the grid resource allocation problem.

5. Designing and development of a new grid resource allocation method.
6. Performing a computer simulation to validate the grid resource allocation method.
7. Performing a comparative performance analysis of the new grid resource allocation method with other well known methods using synthetic and real workload traces.
8. Formulating the mathematical model for the grid job scheduling problem.
9. Designing and developing new grid job scheduling algorithms.
10. Performing computer simulation to validate the grid job scheduling algorithms.
11. Conducting a comparative performance analysis of new the grid job scheduling algorithms with other well known algorithms on an experimental grid using synthetic and real workload traces.
12. Conclude from the above steps.

The flow of the research activities is presented in Figure 1.1.

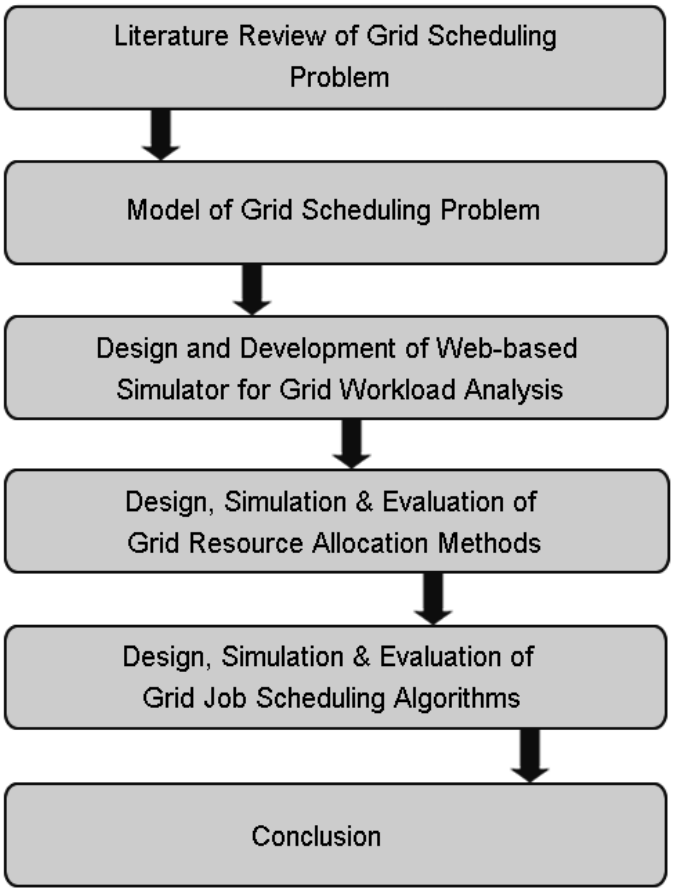

Figure 1.1: The flow of the research activities

1.8 Research Scope

This thesis focuses on improvement of some existing resource allocation and job scheduling algorithms, towards the ultimate goal of enhancing the performance of grid. Prime scopes of this research are as follows:

1. Designing and development of grid scheduling model for dynamic grid scheduling environment. This scheduling model provides the foundation for designing, development and evaluation of resource allocation and job scheduling algorithms. To develop this model, the basis assumption is that, the resources are already discovered. Here, the resource signifies processor and bandwidth only.

2. Designing and development of a web-based simulator for grid workload analysis. The web-based simulator provides a comprehensive characterization of grid workload traces. Grid workload traces have been used for evaluation of resource allocation and job scheduling algorithms. The applicability of this simulator is limited to different workload traces in grid workload format (GWF) only.

3. Designing, development and evaluation of the new resource allocation method for an optimum utilization of resources. Software has been developed for comparative performance analysis of various grid resource allocation methods using synthetic and real grid workload traces, but yet to be tested for real life applications. Here, to evaluate the performance of resource allocation methods, the “computational cost” has been considered as the only parameter.

4. Designing, development and evaluation of the new job scheduling algorithms for efficient and effective execution of jobs. From the more practical perspective, proposed algorithms have been evaluated by comparing with other well known scheduling algorithms for various scheduling performance parameters on an experimental computational grid under dynamic grid scheduling environment using synthetic and real workload traces. The goal is to minimize the average waiting time, average turnaround time, average response time, average bounded slowdown time, maximum total completion

time and maximum stretch time. Though the above algorithms have been designed for grid environment, but the load distribution by these algorithms has been done in a centralized manner, not in a truly distributed way. Also the performances of the algorithm are yet to be tested for real life applications.

1.9 Structure of Thesis

This thesis is structured into seven chapters. Figure 1.2 represents organization of thesis.

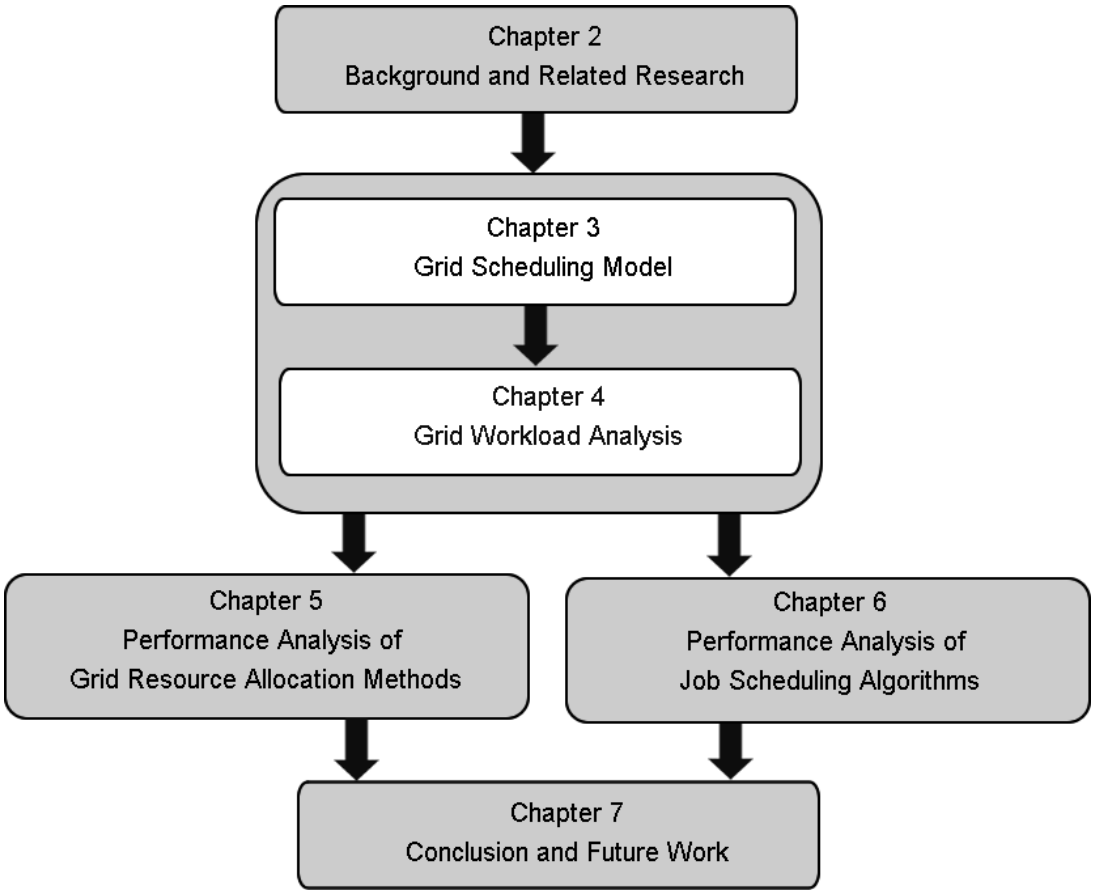

Figure 1.2: Thesis organization

Chapter 1 begins by introducing the whole research and also is added with a brief background on all concepts involved in this work, motivation of the novel approach, the problem statement, research question, objectives, research scope, and relevant research activities.

An overview of background study of various computing architecture, grid computing and grid scheduling are presented in Chapter 2 in which several related works and review are covered. This chapter includes types of grids, the resource types in grids, background of the scheduling problems, the types of scheduling algorithms, and the computational models for grid scheduling. This chapter also focuses on the recent approaches of job scheduling and resource allocation for solving scheduling issues in grid computing. Finally, grid system performance and optimization criteria have been highlighted in this chapter.

The grid scheduling model and proposed scheduling algorithms are elaborated in Chapter 3. This chapter introduces a grid scheduling model with some basic assumptions and presents the linear programming model for grid resource allocation. This chapter proposes new algorithms for resource allocation and job scheduling. This chapter also includes the performance metrics; which will be considered for evaluation of scheduling algorithms. Proposed performance evaluation strategy based on the benchmarks is also included in this chapter.

Chapter 4 presents the need of workload analysis tool. This chapter is concentrated in a discussion about development of a web based simulator (i.e., SyedWSim) and statistical analysis of grid workload traces.

Chapter 5 describes the comparative performance analysis of the new method for grid resource allocation with other well-known grid resource allocation methods using simulation. Theoretical performance analysis of the proposed and various grid resource allocation methods are also thoroughly explained in this chapter. This chapter also presents a new simulator which has been developed to produce a comprehensive simulation of a number of grid resource allocation methods.

Chapter 6 presents a comparative performance analysis of proposed scheduling algorithms with other various job scheduling algorithms. This chapter includes an extensive experimentation for evaluation of scheduling algorithms on an experimental grid using synthetic and real grid workload traces; taken from leading computational centers. This chapter also includes the homogenous implementation of new as well as other scheduling algorithms. The scheduling simulator’s design and development is

also discussed in this chapter. The detailed performance analysis of new and other job scheduling algorithms is also presented in this chapter.

Finally, Chapter 7 concludes thesis with a discussion of the main contributions in this and future research directions in the area of grid scheduling.

1.10 Chapter Summary

This chapter presents the motivation, research problems, research questions, research objectives, methodologies as well as activities, research scope and structure of thesis. The main issue with grid computing is to manage the resources properly and schedule the jobs more efficiently and effectively. The main aim of this research is to design and evaluate new algorithms for efficient resource allocation and job scheduling on computational grid.

CHAPTER 2

BACKGROUND AND RELATED RESEARCH

2.1 Chapter Overview

This chapter presents the background study of various computing architecture, grid computing and grid scheduling. It is organized as follows: Section 2.2 presents various computing architecture for high performance and section 2.3 shows the need for grid computing systems. Characteristics and architecture of grids are discussed in section 2.4 and section 2.5 respectively. Section 2.6 discusses the types of grids followed by the explanation about the resource types in grids presented in section 2.7. Section 2.8 then explains the scheduling problems in detail. Section 2.9 discusses the types of scheduling in a grid. Section 2.10 is about the computational models for grid scheduling. Section 2.11 focuses on the algorithms of job scheduling and resource allocation for solving scheduling issues in grid computing. This chapter is concluded by highlighting grid system performance and optimization criteria as presented in section 2.12.

2.2 Parallel and Distributed Computing Architectures

Problems associated with scheduling on distributed environments and parallel machines become more understandable with better knowledge of the advancement of parallel and distributed architectures.

This section provides a brief description of present architectures, moving through the simple multiprocessors computers to supercomputers resulting in distributed environments such as grids. Every kind of architectures extends another one to

provide more computing power. The hierarchical structure of the different platforms is presented in these following sections.

2.2.1 Multiprocessor Computers

The very first family of parallel machines is Symmetric Multiprocessing (SMP) computers [35]. A computer enters into this family if each of the computing elements shares the exactly identical data utilizing a single main memory. Most modern operating systems can easily take advantage of these structures [36]. Figure 2.1 below presents the architecture of a SMP computer.

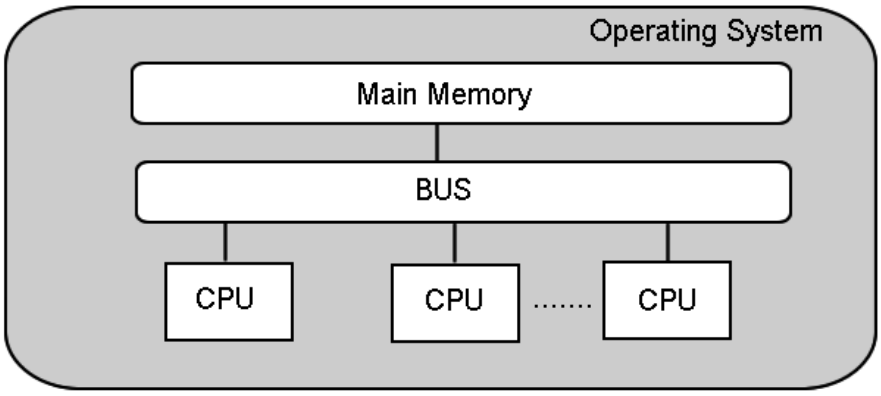

Figure 2.1: The SMP architecture

SMP architectures are very well designed to run the small-scale parallel applications. Each thread or process can run using a processing element although all processes have an access to similar data from the main memory. The application not only takes advantages of the parallelism but also avoids the context switching inherent to the execution of a number of applications in mono-processor architectures. However, programming on SMP architecture requires more programming paradigms [37] than just a very simple sequential development and thus helps to make the development more challenging. Higher-level APIs such as OpenMP [38] are aiming at reducing the difficulty of programming of these kinds of platforms.

Having an individual memory space makes it possible for very easy data sharing among the computing elements. However, as compared to the memory, the processing elements are capable of processing, storing and distributing data more rapidly.

With the development in processing speed, the memory accesses have been unable to keep up with the demands of the computing elements consequently leading the number of processors on such platforms to be limited in use. To cope with this bandwidth constraint, new architectures called Non-Uniform Memory Access (NUMA) computers were designed. In practice, the bandwidth required by the memory and the buses to send the data quickly adequate for the processors is not achievable, thus NUMA features distributed memory access [39]. Several memory banks are hierarchically connected to various processors, yet every single processor can gain access to all memories through a fast communication link. Figure 2.2 schematizes this distributed memory.

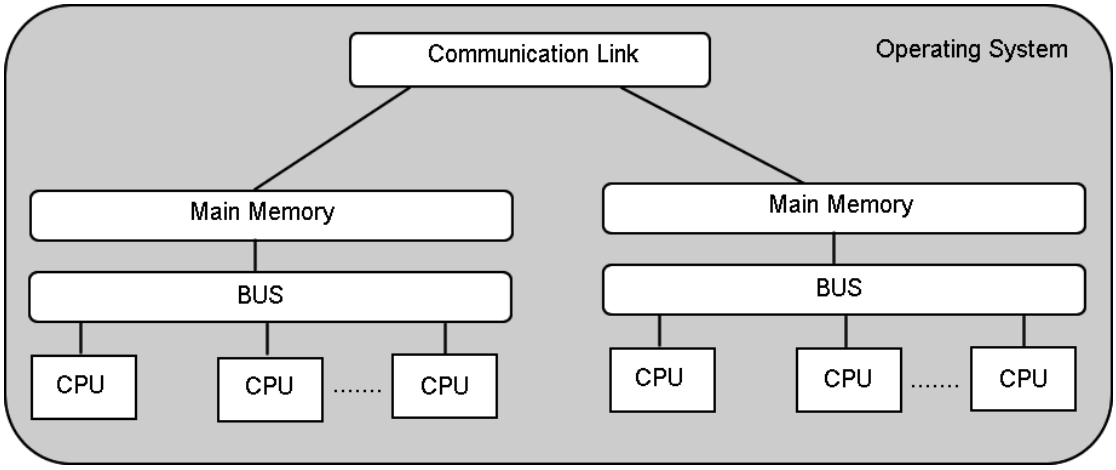

Figure 2.2: The NUMA architecture

Even though all memories are available for every process, the amount of time needed to collect the data locally could immensely vary depending on the location of the data themselves. Moreover, if another memory bank holds the data, this could give rise to latencies.

NUMA architectures are possibly harder to program than SMP in that a user could localize the data in the memory by him/herself to gain a maximum performance [40] thereby increasing the difficulty in programming. However, since NUMA architectures are supported by Linux 1, Windows 2 and Solaris 3 - the main operating systems, it is still possible for applications to be programmed as if they were on an SMP computer but without the advantage of localizing the data in the memory.

2.2.2 Clusters

When a set of various computers are joined closely to work together, it is known as a cluster of computers [41], [42], each of which in that cluster individually is called as node and controlled by a single operating system, which in fact might be unaware that the node belongs to the cluster. A node in a cluster is usually an SMP machine. A common example of a cluster is shown in Figure 2.3.

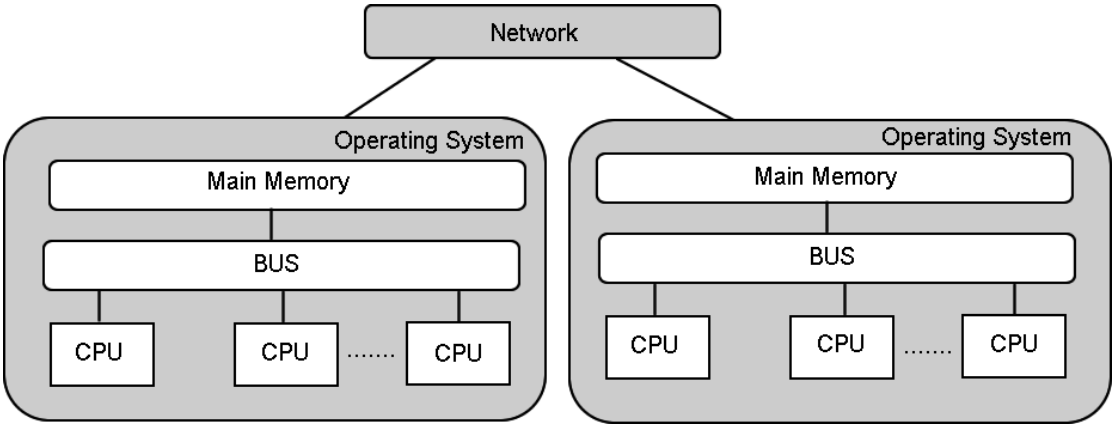

Figure 2.3: Architecture of a cluster

Clusters do not have a central memory, which is easily accessed by all the processes, yet some clusters use distributed file systems such as NFS. However, this type of solution, due to its tardiness, might cause too much delay. As far as the application is concerned, access to the memory among the various nodes is fully determined by the programmers involved in programming the parallel applications. For example, the most popular method that programmers use to send data between nodes is the Message Passing Interface (MPI) [43], a basic API, providing a variety of communication methods for sending and receiving data across various processes spread out among the computing nodes. The Parallel Virtual Machine (PVM) [44] in addition is another notable communication library.

When some independent computers are joined together, they can form a less expensive, more expandable, and more reliable cluster than one server handling multiple processes. In fact, the nodes in a cluster can still continue working without any interruption, even when one node stops functioning. Moreover, since each node is unaware of the others, a user could expand the system with as many other nodes as

user desires; at this point a cluster is more expandable than a single computer. Several thousands of computing nodes could possible make up one cluster [41],[42].

2.2.3 Supercomputers

Supercomputers firstly developed in the 80's to perform processing procedures on a large scale nowadays are at the forefront of technological advances. Those firstly were installed with anywhere from four to sixteen processors, and until recently there has been an increase in the number of processors installed in the computers. Today a supercomputer could have as many as a few hundred of thousand cores. The Top500 contains the basic standing of supercomputers [2]. The standing of all systems is on the basis of speed, meaning that systems are ranked according to how fast they can run the standard application known as LINPAC, which was a created in order to find solutions for a system of full of linear equations. Supercomputers receive their standing in regards to the floating point operations per second (FLOPS) that they achieve. In June 2010, the Cray Jaguar with 224,256 cores was considered to be the fastest computer achieving a top performance at 1.759 PFLOPS [2], [3].

The 2011 International Supercomputing Conference in Hamburg in June 2011 introduced a new standing for supercomputers in which the K computer system was put at the top of the TOP500 list. Intriguingly, the K computer system, even though not completed yet, is even more powerful than a combination of the five listed systems coming after it. It, while being in the building stage still, has nearly double the number of cores as any other systems listed in the TOP500. Presently, it has 68,544 CPUs containing eight cores, a total of 548,352 cores, all held in 672 computer racks. Moreover, this unfinished system obtained the best LINPAC standard performance in the world at 8.162 petaflops (quadrillion floating-point operations per second); making it to be the TOP500's number one [2], [3].

Besides, it also has a 93.0% ratio of computing efficiency, which is registered as a superior standard for systems. Japan's Ministry of Education, Culture, Sports, Science and Technology (MEXT) introduced the High-Performance Computing Infrastructure (HPCI), which the K computer is a part of. Furthermore, its name comes from "Kei",

which in Japanese means 10^16 (ten quadrillions); standing for the aim of the computer which is to perform at a rate of 10 petaflops. The K Computer can be found at the RIKEN Advanced Institute for Computational Science (AICS) in Kobe. RIKEN is the Institute for Physical and Chemical Research [2], [3]. The layout of the K Computer is presented in Figure 2.4 below.

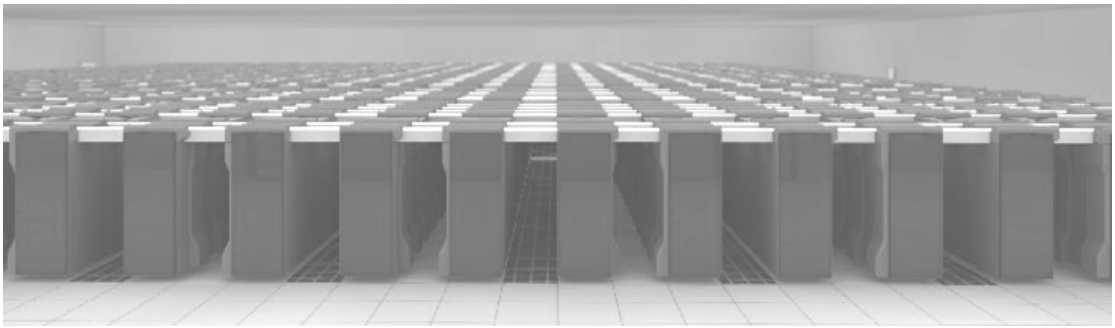

Figure 2.4: K Computer [2]

The clusters of today's supercomputers are very finely tuned and joined together very closely enabling them to handle the huge number of processors involved. Communication plays an extremely important role in these types of computers; thus, to obtain the highest performance possible, dispersion of network topologies are required [3].

Linux, obviously presented in Figure 2.5 taken from [10], is the most common operating system found in all the operating systems of the supercomputers in the Top500. It can be seen that from 1998 more than 90% of the market has been controlled by Linux.

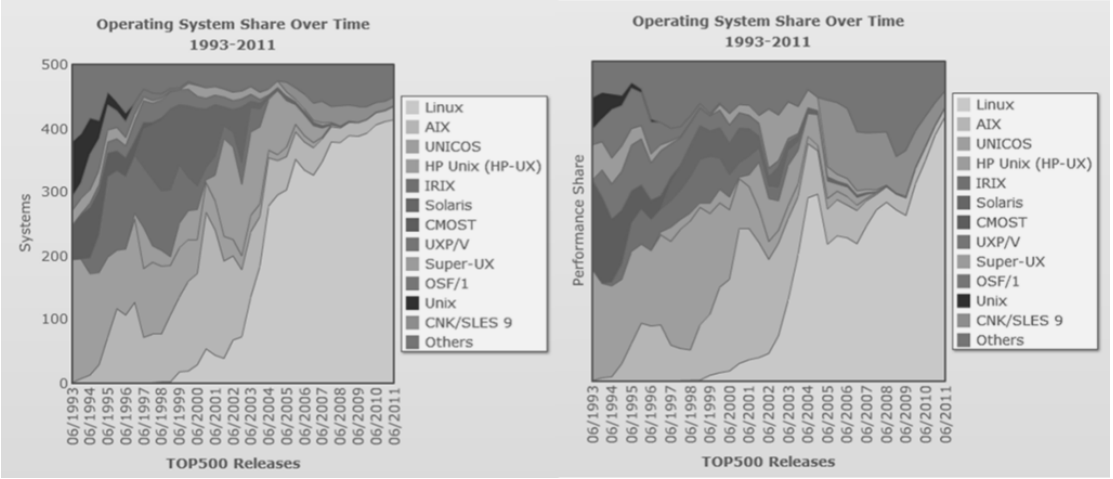

Figure 2.5: Operating systems used in the Top500 [2]

Even though supercomputers, due to their ability to find the solutions for big problems, could provide very well benefit, the cost is still unaffordable for small and medium sized businesses. The expense of the hardware itself is not the main issue, but the system maintenance is the one. A supercomputer commonly is to be housed in an entire building designed just for that purpose. A suitable construction for this must have an adequate amount of energy to continuously run the computer system, while overheating simultaneously must be controlled by utilizing the proper infrastructure necessary to keep such a structure cool. As a result of such the huge costs required, supercomputers then are only affordable for very big businesses or organizations [3].

2.2.4 Grid Computing

Distributed and heterogeneous resources need to be collected if the organizations lacking for the processing resources are going to be able to use the huge processing power of supercomputers. When a collection of individual systems is presented to the users as a singular and incorporated system, it is known as a 'distributed system'. In such a system, the software and hardware elements of the joined computers talk to each other and organize their actions by message passing [45].

In [1], Foster and Kesselman describe the meaning of 'computational grid' to present the distributed computing architecture. An electrical power grid, as its name implies, is as with the power grid that everyone can independently access the grid even though the actual source is unknown. In computational grids, instead of electric power being accessed, processing power, storage and such are the accessed resources.

Foster and Kesselman firstly introduced the grid by defining it as "a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities." The framework for the supercomputer belongs to a variety of combined organizations but none of the company entirely has a complete control over the grid[1].

Their first description introduced the grid as a collection of resources owned by a set of different groups, yet nothing was presented to clarify the guidelines on how these groups would share the resources they mentioned. Hence, the authors added to

their description in [21] by introducing the idea of Virtual Organization (VO) in handling this specific matter: “grid computing is concerned with coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations.” Active, multi-group virtual organizations face large scale issues for which it is vital to find solutions for; computing by grid in turn allows for the sharing, choosing and gathering of geographical resources in order to obtain these solutions.

Since the resources involved in these systems are spread over a large geographical area, the grid gives a resizable VO that allows the resources available to be easily shared among multiple individuals or organizations. A VO can be considered as a domain; however, no control, centralized point or relationships of trust also indicate a lack of knowledge regarding the combined systems. The VO does, however, have the usual objectives it must constantly follow. The primary components of a VO are the information providers and service providers such as application, storage and CPU cycle servers as well as the individual users of the system. In fact, a VO is not an actual environment such as an office and can be considered more a network[21].

Foster gives a basic description of the three primary elements of architecture that make it to be a grid [46]. A computational grid is a system, which:

1. Organizes resources, which are not centrally controlled.
2. Uses common, open, general-purpose protocols and interfaces.
3. Delivers nontrivial qualities of service.

Grids in the High Performance Computing (HPC) society usually contain multiple clusters [47], [48].

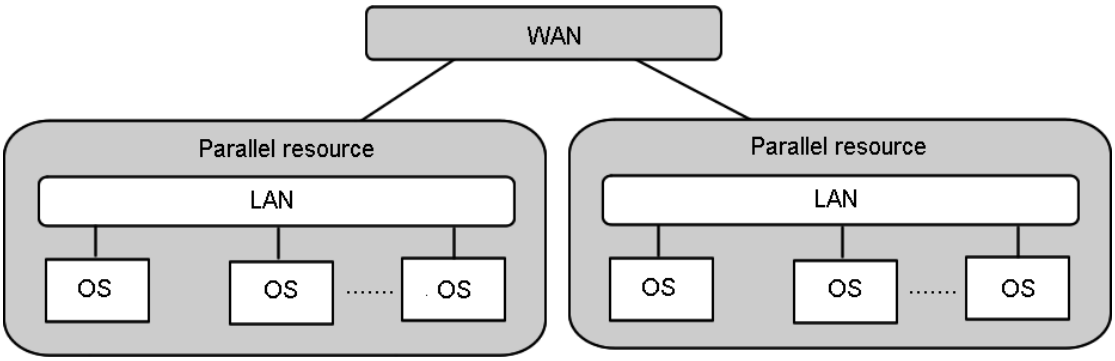

Figure 2.6: The multi-cluster grid architecture

As shown in Figure 2.6, multi-cluster systems are parallel machines or clusters that are joined by fast networks covering a broad area. In a situation where organizations are held within a single VO, it is normal for individuals in the group to have one or more of their own clusters available for the group's use. Furthermore, it is possible that various software and scheduling policies to be used to handle individual clusters. However, resources are always openly and clearly accessed; basic Application Programming Interfaces (APIs) or some types of middleware are used for this purpose. In this way, in regards to the three conditions mentioned above, a grid which contains multiple clusters already fits the criteria. While the application of a multi-cluster grid in the HPC is very common; grids of other types are also used [47], [48].

2.3 The need for Grid Computing Systems

A need for more and more processing power for both research and business seems to be endless even though the available processing power has been increasing at an astonishing speed for quite some time. This is especially true by considering the potential new projects in both areas of science and business, which recently will need immense processing power. The rate of increase in network bandwidth is growing faster at a rate than that of processor speed; thus making logical to join multiple computers that can use processing power efficiently [46]. At present, grid computing is deemed to be the most efficient method to do this.

An article published in The New York Times claimed that "All Science Is Computer Science" [49]; this statement was made as works done in various areas of science such as biology and physics depend on simulations which are continually complicated. The need for greater processing power hence comes to be more relevant than before. While greater vision is still important to emerge with fresh ideas for work in these fields, as far as processing power is concerned, any experiment needing to be performed are still limited. Grid computing is vital as far as technology for processing can aid in advancing science in all fields.

The prevailing technology used for resource sharing on large scale is grid computing [21]. It raises the processing ability of a system; even is often used to find solutions for scientific big and complicated problems and processing resources which are spread out according to geography. Research and development of the grid computing is slowly but steadily increasing in that many large-scale, complicated scientific problems are still unsolvable using common networks. A grid computing system connects the available computing resources, such as computers, applications, and storages devices to attain the high processing and lower the processing time of applications [1], [50].

Grid computing over the past few years has played a vital role in achieving advancements in computing-intensive areas such as medicine, physics and meteorology. Collaborative/e-Science Computing [51], [52] and Data-Intensive Computing [53] are good examples of computing infrastructure, and well known for their ability to provide optimized processing [18], [54], [55], [56], [57].

2.4 Characteristics of a Grid

It is vital to understand the grid's characteristics in order to create a more appropriate grid system because in many ways, it is different from the traditional computing infrastructure most commonly used. The following sub sections present the characteristics of a grid.

2.4.1 Distribution and Sharing

Distribution [15], [58] is one of the most significant characteristics of the grid. It is related to the various resources that could be databases, computers, digital libraries and other scientific tools in various geographical locations.

Grids, not centralized but rather distributed, suffer from issues involving computation for this particular feature. This problem then requires the grid's management system to solve issues involving control of resources, scheduling of jobs,

security transmission, use of systems in real-time and the possibility that some forms of intervention may be needed by the users [14].

Resources throughout the grid, while being well distributed, are also completely shareable; this means that any user becoming a part of the grid has a complete access to all the resources of the grid. The arrangement of sharing resources is the base concept of the grid system as represented by the statement, “Sharing is the purpose of the grid and without it the grid is meaningless” [21], [59]. This concept is comprehensive and allows for a computer in one place not only to be able to complete a job at some distant locations, but also to allow all the computers to share data such as models, databases and results as they are processed.

Furthermore, with the support of the management system of the grid, the physical feature is the distribution while the logical one refers to the implementation of sharing.

2.4.2 Self-similarity

Self-similarity [1], [60] is present extensively in social as well as in natural phenomena. Similar to the grid, almost all systems of a complex nature have several special features. Components in the local part of the grid are quite equal to those in the global part of the grid; thus both the global and local areas possess the similar features. The idea of recursion expresses this to a certain degree.

2.4.3 Dynamic and diversified

Today’s grid structure is dynamic meaning that certain resources once present in the grid might not be present any longer or even suddenly stop working. Furthermore, resources previously not in the grid might become a part of the grid later. Dynamism in this situation is the increase and decrease of resources.

The grid’s resources are both heterogeneous and diversified. For this the grid system must solve problems such as communications between different operating systems.

2.4.4 Self-manageable

The resources in a grid are owned by a particular organization or user; therefore the highest administration rights belong to the owner. The grid itself cannot have the direct control over the resources but can manage the resources.

2.5 Grid Architecture

There are five basic elements, which make up the architecture of the protocol for the grid [21]. It has layered architecture similar to Internet protocol architecture. Figure 2.7 presents a diagram of the two architectures side by side.

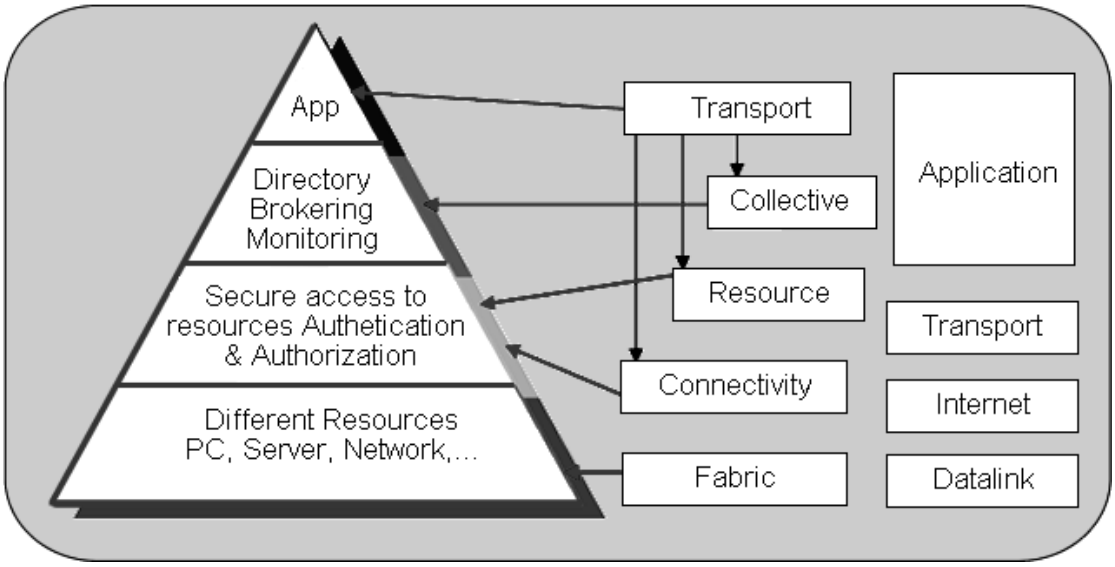

Figure 2.7: The layered grid architecture and its relationship to the internet protocol architecture adapted from [21]

The following are the five layers of grid architecture, which are described in the following sub sections.

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

2.5.1 Grid Fabric Layer

The fabric layer is made up by the resource-specific and site-specific components and possesses low-end and high-end computers, which include networks and clusters, as well as scientific tools, and resource management mechanisms. Examples of those components could possibly include a support for interfaces and advanced reservation, which helps with services at higher levels when they need to assign (co-schedule) resources in an interesting manner with a possibility to obtain. Examples of these mechanisms could possibly include resource management, interfaces, and advanced reservation that would possibly help to provide higher-level services to aggregate and co-scheduling resources in interesting ways that would be impossible to achieve the network's quality of service in some routers. Fabric layer also provides administer sharing activities for resources at higher levels [21]

2.5.2 Grid Connectivity Layer

This layer deals with authenticating and authorizing protocols and core communication protocols for grid-specific network transactions. The resources of the fabric layer utilize these communication protocols for data exchange. Some elements of security are also provided by the connectivity layer including delegation, single sign on and the user-based trust relationship [21].

2.5.3 Grid Resource Layer

The connectivity layer is expanded upon by the resource layer, which assigns the protocols (APIs and SDKs) in regards to securing the negotiation, initiation, control monitoring, accounting, and payment of sharing operations related to individual resources. Moreover, it provides management of remote processes, co-allocation of resources, access to storage, information security, and Quality of Service such as reservation and trading of resources [21], [61].

2.5.4 Grid Collective Layer

The focus of the collective layer is on protocol and services such as APIs and SDKs; these are all associated with resource collections. Directory services are facilities that the users can use to search for resources and current load that are readily accessible. Services such as co-allocation, brokering and scheduling allows the users to ask for resources to schedule their jobs from among the suitable resources, which are at hand. Some facilities are to monitor resource or network failures, current resource load and overload, and provide intrusion detection etc; known as monitoring and diagnostics services. Reducing response time and cost while optimizing reliability comes to be possible since data replication services copy data thus making the performance of data accessing to be improved [21].

2.5.5 Grid Application Layer

The application layer, the last layer in the grid architecture, is made up of users' applications, which have been created 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 [21].

2.6 Types of Grid

Grid systems can be differently categorized as follows:

2.6.1 Traditional Grid

It is a closed network of computers providing its access only to a few consumers and can be managed by a single administrator. Traditional grids[62] for instance are often built for a specific purposes such as NASA information power grid [63] - established for research that is only open to scientists and engineers working for NASA. Traditional grids are mostly of homogenous nature and can offer maximum performance for their single user ownership. On the other hand, these grid systems have minimal flexibility for being built for specific purposes.

2.6.2 Computational Grid

It consists of resources that are explicitly designed to achieve high computing power and has mostly high-performance computing server. A computational grid is a system that provides higher aggregated computational power than any single personal machine. According to the usage of the computing power, computational grids have been further divided into two subcategories: distributed supercomputing and high throughput. A distributed supercomputing grid makes use of the parallel execution of applications over multiple machines simultaneously to minimize the overall execution time. On the other hand, the goal of the high throughput grid is to maximize the completion rate of a set of jobs while utilizing available idle computing cycles as much as possible [14], [18], [57], [58].

Computational grids have been designed with the objective of maximizing the computational power enabling to run complex scientific application through the sharing [18], [57].

2.6.3 Data Grid

The purpose of the data grids is to provide large scale computing infrastructure to the next generation applications, which will support high demanding computation and analysis of shared databases across widely distributed scientific communities [14], [18], [57], [58].

Data grids not only deal with large-scale data repositories, access, sharing but also deal with large amounts of distributed data belonging to different organizations. Grid data are stored in different locations in which the consumer are not concerned where the data are and how they can access it. For example, more than two hospitals working on a heart disease research require a large amount of data. For this, they can build data grid and share data by this way. In such types of grid, a number of algorithms have been designed to maximize the performance and efficiency of grid-enabled applications. In addition, data copy and transfer are key procedures which help to attain high computing throughput [18], [57].

2.6.4 Storage Grid

A storage grid offers a mechanism to combine the spare storage resources in grid environments and provides services like users transparent and secure storage[64]. For example, Network Attached Storage (NAS) and Storage Area Network (SAN) provide shared storage for a number of servers and multiple protocols, and furthermore, more than 30 terabytes of Premium and Enhanced storage have been provided by the UC Berkeley Storage Area Network (SAN) to more than 20 clients and 100 TB expected in future. Advantages of SAN technology are to enhance availability, maximum performance, and better monitoring through centralized administration[65].

2.6.5 Peer to Peer Grid (P2P-G)

As well as on grid technologies, P2P-G is based on Peer to Peer - a resource sharing method available at the edge of the internet through ad hoc overlay networks by means of symmetric communication. A P2P computer network is determined by the computing power and bandwidth of the computing nodes participating in the network. In P2P, instead of creating a large scale network, like the internet, one can be directly connected to the specific system that can provide the desired computing power, thus an overhead could be prevented. Only computers running the same type of software can be joined to meet their demands. P2P is supposed to be a variant of data grids as its aim is also for data exchange. Tasks are allocated to grid nodes in a decentralized way [66]. P2P grid furthermore possesses the properties of reliability and robustness and is widely used in many cases [67].

2.6.6 E-Science Grids

E-Science grids are built to solve various emerging problems of science and engineering by providing support to the computational infrastructure. UK e-Science grid, EGEE grid computing, German D-grid, the Dutch e-Science grid and French Grid'5000 are several representative examples of e-Science grids [18], [57], [68].

2.6.7 Enterprise Grids

At present, grid computing has become an important component of business. E-business should be able to satisfy the growing needs of consumers. It should have the capability to adjust itself with marketplace dynamically and efficiently [18], [57].

Furthermore, enterprise grids share resources transparently and enable execution of several projects at large scale enterprise. However, the great and innovative issues on how computing power is used appear in enterprise grids. These grids posses high potential to solve business issues by providing global access to enterprise computing services and data [18], [57], [69]. Popular examples of enterprise grids are IBM grid, Sun grid engine, Oracle grid and HP grid.

2.6.8 Desktop Grids

It is a new type of enterprise grids consisted of hundreds or thousands of desktop machines. It provides high processing power by using idle cycles of desktop machines in small enterprises or institutions. A number of idle machines can be used to setup a Desktop grid for the small scale institution. This grid is very easy to build, and unique from administrative perspective. It is easier to control due to less volatility and heterogeneity of resources [18], [57], [70], [71].

2.7 Types of Resource in a Grid

Various types of resource available in the grid include computation, databases, data and storage, special equipment, software and licenses, communications links, capacities, architectures, and policies [18], [21], [57].

2.7.1 Computational Resources

The most important and common resource in the grid, Computational resources can be various in architecture, software platform, speed and connectivity. They allow CPU scavenging to make good utilization of resources. When any computer gets free

and idle, it then notifies its state to the grid. With the help of this, users are motivated to join the grid environment. Computational grid combines the processing power from the distributed computing nodes and also grants the computational power to process complex jobs. Computational resources meet the business requirement for instant access to resources on demand [58]. In the grid system, the use of the computation resources is made by three following major mechanisms[18], [57].

a. Existing parallel applications can run on the grid.
b. Applications or job can be divided into tasks, each of which can execute in parallel fashion on different machines in the grid.
c. Application simultaneously can run several times on various different machines.

2.7.2 Data Storage Resource

The second most common resource in grid, data grid as the secondary storage such as HDD and type driver is to increase capacity, or memory attached to the processor, performance, sharing and reliability of data. Data grid grants an access to the datasets as well as scalable storage. Catalogued, Replicated, as well as even diverse datasets are being stored in different positions for data grid to create an illusion of mass storage. Using a unified file system such as Andrew File System (AFS) and Network File System (NFS) with the storage on multiple machines will increase the capacity. These advanced file systems can duplicate sets of data. An intelligent grid scheduler meanwhile can help to choose a suitable storage device to hold the data depending on usage job patterns. Furthermore, in grid method, journaling can be implemented by grid file system as a result of a more reliable data recovery after the failures. Then the data are shared and updated by plenty of users, and grid file system executes advanced synchronization mechanism to decrease contention between these users. Also data striping in writing or reading consecutive records to/from different physical devices overlap the access for faster throughput [14], [18], [57].

2.7.3 Communication

It is 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 VPNs are needed to overcome potential network failures as well as huge data traffic. In an inter-grid, by assuming that a search engine is going to be developed and should access the external Internet to provide connectivity among the grid machines; these connections in this case will add the new total available bandwidth for accessing the Internet rather than sharing the same communications path [18], [57],[72].

Meanwhile, software and licenses, architecture, capacities, special equipment as well as policies characterize a different kind of resources. The cost for installation of too expensive software on each grid machine can increase. To avoid this, this software is installed on some particular machines in which jobs require this software to be sent [18], [57].

This method then can reduce the cost for an organization. On the other hand, some software licensing arrangements permit 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 [18], [57], [58].

Because of the heterogeneous as well as the dynamic nature of the grid, it often has different operating systems, capacities, devices, architectures, and policies. Each item stands for a different kind of resource while the grid allocates jobs to machines since it can use this special equipment, architecture, capacities and policies as criteria. For instance, there are numerous types of software running on different architectures such as Sun Ultra, SGI origin, x86, etc. At this point, users must consider such characteristics while assigning the jobs to machines in the grid [18], [57], [58].

In general, categories of grid resources are shown in Figure 2.8. A grid resource can comprise of resource ID, resource name, performance criteria, and cost (price) [58].

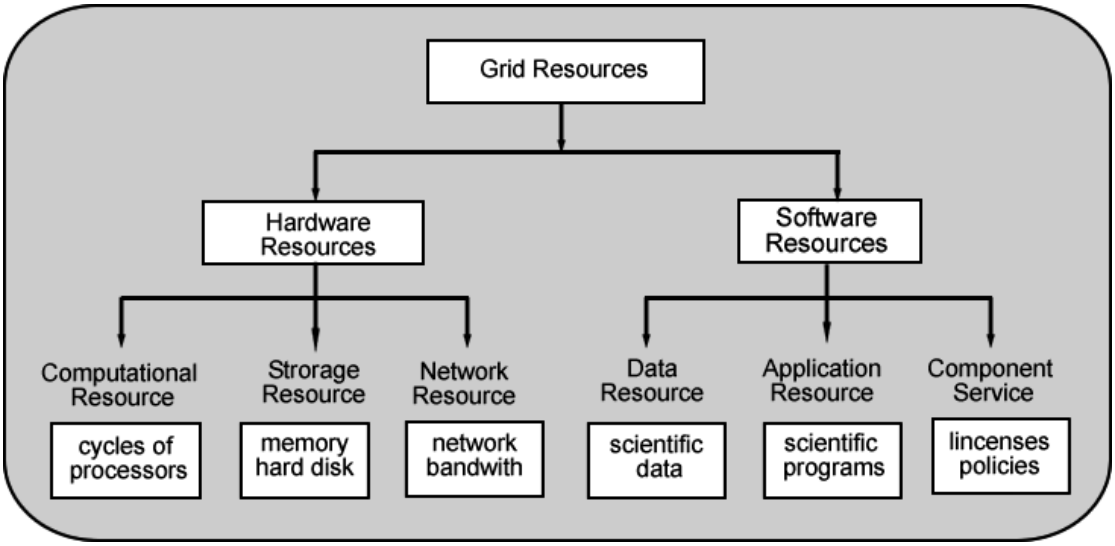

Figure 2.8: A classification of grid resources [58]

2.8 Scheduling problems in grid systems

Grid computing is progressing well but still has many fascinating issues that need to be addressed. Of the challenges, two major considerations that need to improve are performance and efficiency – those becoming the overall aims of grid scheduling as a very important high computational mechanism for the efficient and effective scheduling of the applications and optimization of the utilization of resources in the grid [18], [57], [58].

As a core component of grid infrastructure, grid scheduling is responsible for an efficient and effective utilization of heterogeneous and distributed resources. Grid scheduling is a process of ordering both tasks on computing (or "computer") resources, and communication between them – also known as the allocation of computation and communication over time [18], [27], [57], [58].

Due to the very dynamic and unpredictable nature of grid resources, scheduling is becoming very challenging in grid computing. Its problem can be viewed as a multivariate optimization problem, where the application being assigned to a set of machines is to optimize the overall execution time. The job scheduling problem is an NP-Complete problem [18], [26], [29], [73], [74], [75].

2.8.1 Characteristics of grid scheduling

Grid scheduling problem is one of the most researched issues in optimization domain. Many characteristics in the grid environment however make the challenge to be different and more demanding than traditional distributed systems. The followings are a few of these characteristics [18], [57].

2.8.1.1 The dynamic structure of the computational grid

Unlike conventional distributed systems, resources in a grid system can enter or leave the grid unpredictably. This dynamicity occurs due to connection failure with the system or machine is turned off, or the operating system is modified, and so forth. As long as the resources cross various administrative domains, there is absolutely no control over the resources[18], [57].

2.8.1.2 The high heterogeneity of resources

A grid system consists of number of computational resources of various processing capabilities. These computing resources could possibly become available from desktops, laptops, clusters, and supercomputers. Now a day's grid infrastructures are not so adaptable and flexible but heterogeneity is the most important characteristics in grid [18], [57].

2.8.1.3 The high heterogeneity of jobs

Jobs arriving at any grid system are diverse and heterogeneous when considering their computational demands. Some jobs might be demanding high computing power for their processing whilst others may require few units of processing to meet their demands. In truly dynamic environment, the grid system is not aware of jobs arriving in the system [18], [57].

2.8.1.4 The high heterogeneity of interconnection networks

A number of participating computers in grid are linked together using different network topologies. Transmission cost is an important parameter to measure the performance of grid. Thus, the heterogeneity of interconnection networks is also necessary to consider in design of grid [18], [57].

2.8.1.5 The existence of local schedulers

Grids are expected to be constructed by the ``contribution" of computational resources throughout organizations, universities, companies and individuals. Many of these resources could ultimately be managing local applications and use their local schedulers such as Condor system. In these cases, a single possible requirement is usually to use the local scheduler [18], [57].

2.8.1.6 The existence of local policies on resources

As the grid resources are owned by different organization; that's why an individual can't fully control the resources of grid. Computing demands of companies or individuals are changing, can't be predicted. In some cases, companies would like to minimize the contribution of their resources to grid. There is need for planning to make use of resources effectively, like access of resources, pay- per -use, available storage, etc. [18], [57].

2.8.1.7 The job-resource requirements

Present grid schedulers consider full availability and compatibility of resources while doing scheduling. In real situations, numerous restrictions and incompatibilities could possibly be base on job and resource specifications [18], [57].

2.8.1.8 The large scale of the grid system

Grid system is large scale computing infrastructure and dynamic in nature. Many applications, tasks or jobs are joining the grid system over time. Consequently, there is need to design and develop a grid scheduler to make an efficient execution of jobs and effective management of resources. A number of attempts have been made to propose different types of schedulers to meet the growing demands of applications, and support scalability [18], [57].

2.8.1.9 Security

Security is also one of demanding issue in grid scheduling [58]. Security can be observed from two perspectives. Firstly, task, application or job could have their own security demands. Secondly, computing node might have its own security demands. When one application is running on one node; then other grid nodes could not see it [18], [57], [58].

2.8.2 A general definition and terminology

A grid scheduler can be defined in different ways depending on the organization as well as the characteristics of a grid system. In a scenario, a grid scheduler is actually running permanently as follows: receiving new jobs from users, looking into the available resources from the grid, selecting the most likely resources in line with the performance criteria, availability, reliability and performance. Then, finally grid scheduler generates a mapping of jobs to the selected resources. The following terminology have been introduced for scheduling in the grid systems [18], [29], [57], [58].

- **Job:** A job is defined as a set of tasks with different processing demands and has different requirements of resources like number of CPUs, memory, number of nodes, software libraries, etc. A job is also linked with a set of constraints. A job may has one task in the most simplest situation [14], [18], [29], [57].

- **Application:** An application is a computer program to solve a computational problem in grid environment. It might require splitting of the computation into jobs; and then jobs are assigned to different computational nodes in a grid. Application is specified by the various computational resources and set of constraints; which are specified in the application description [14], [18], [29], [57].

- **Task:** It represents a computational unit (typically a program and possibly related data) running on a grid node. However, a unique definition of its concept is absolutely not found in literature. It is usually known as an indivisible schedulable unit. Tasks might be categorized into dependent tasks (grid workflows) and independent tasks [14], [18], [29], [57].

- **Resource:** A resource is a fundamental entity for the computation. Grid scheduler assigns and processes applications, jobs or tasks on resources. Resources have their unique features such as memory, CPU characteristics, software, etc. Some of the attributes of resources change over time like processing speed and workload. In addition, resources might belong to different administrative domains. That's why various policies have been implied on usage and access of resources [14], [18], [29], [57].

- **Specifications:** Task, job and application demands are generally specified using high-level specification languages. Likewise, the resource characteristics are shown employing specification languages. One such language would be the ClassAds language [14], [18], [29], [57].

- **Resource pre-reservation**: Pre-reservation is a well known mechanism in grid scheduling. It is demanded in two situations; firstly, when tasks have demands on completion times and secondly, when a lot of dependencies are involved in the execution of the workflow. Negotiation and agreement protocols are the main components of advance reservation, which further involve resource providers and consumers [18], [29], [57].

- **Planning:** A planning would be the assignment of jobs, tasks, or applications on the computational resources [14], [18], [29], [57].

- **Grid scheduler:** According to performance optimization criteria, grid scheduler is responsible for mapping of jobs, tasks or applications to grid resources. Grid schedulers have been categorized into various levels depending upon their functionality in grid like super-schedulers, local schedulers, cluster schedulers, meta-schedulers and enterprise schedulers. The grid scheduler interacts with other parts of grid in systematic way: Grid information system (GIS), local resource management systems and network management systems. All these types of schedulers have their specific but conflicting goals. Different scheduler interact and coordinate with each other in order to run the job, task or application in seamless manner [14], [18], [29], [57].

- **Super scheduler:** This scheduler is responsible for centralized scheduling by which local job schedulers are utilized for reservation and allocation of resources in the grid environment. Local schedulers have their own queues and manage the execution of jobs at resource level. The super scheduler manages the key activities like service level agreement, advance reservation and negotiation [18], [57].

- **Meta-scheduler:** This scheduler is also known as metabroker. This scheduler is originated when a single job or application is allocated to more than one resource in the grid environment. As compared with super scheduler, it is responsible to make coordination among the local schedulers of the particular machines to compute an overall schedule. Carrying out the load balancing across multiple systems is really a primary objective here [18], [29], [57].

- **Local/ cluster scheduler:** This scheduler is responsible for mapping of jobs to the resources belonging to the same local area network (LAN). As handling the local resources and the local job queuing system, this

scheduler then is a type of ``close to resource" scheduler [14], [18], [29], [57].

- **Enterprise scheduler:** This kind of scheduler comes up in large enterprises having computational resources distributed in several enterprise departments. It utilizes the various local schedulers from the same enterprise [14], [18], [57].

- **Immediate mode scheduling:** In immediate mode scheduling, tasks are immediately planned when they join the grid system [18], [57].

- **Batch mode scheduling:** Tasks in batch mode scheduling are categorized into different batches. Then scheduler makes allocation of these batches to the resources[18], [57].

- **Non-preemptive/preemptive scheduling:** This type of scheduling deals whether an application, job or task can be interrupted during execution or not. In the non-preemptive mode, an application, job or task allocated to a resource, must be completed fully without any interruption at resource level. In other words, the resource cannot be taken back from application, job or task during execution [18], [57]. Preemption is permitted in the preemptive mode scheduling. The current execution of job, task or application can be interrupted depending upon the different criteria like job priorities, resource optimization etc [18], [57].

- **Cooperative scheduling:** In this scheduling, with the synergy of procedures, rules, and grid users, a feasible schedule can be computed [18], [57].

- **High-throughput schedulers:** The core objective of this scheduler is to maximize the throughput of the grid. Throughput is average number of jobs or tasks completed per unit of time. These schedulers are also referred as the task-oriented schedulers because they are mainly focused on task performance requirements [18], [41] ,[57].

- **Resource-oriented schedulers:** Optimizing resource usage is the aim of this kind of scheduler. Hence these schedulers refer to the resource-oriented schedulers, focus of which is within resource performance requirements [18], [57].

- **Application-oriented schedulers:** These schedulers are involved to optimize the scheduling of jobs in such a way to satisfy a user's performance criteria. In order to attain the most effective performance of applications, these schedulers have to consider the application specificity as well as system details. Another thing, the interaction with the user need be considered as well [18], [57].

2.8.3 Phases of scheduling in grids

The grid scheduler has to follow a sequence of steps in order to carry out the scheduling process. Grid scheduling process can be divided into three main phases (1) Resource Discovery; (2) Resource Allocation; and (3) Job scheduling. In [76], [77], [73], the general architecture of a grid scheduler is described.

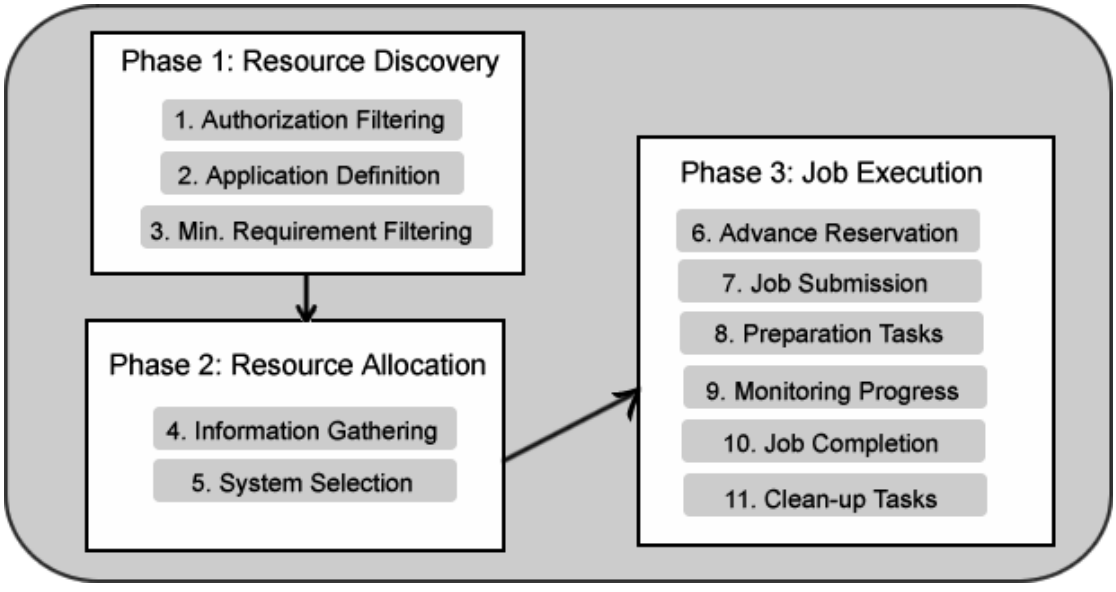

Figure 2.9: Steps of a general grid scheduler [73]

Resource Discovery is an important task within grid resource management system. In fact, in realistic grid applications, it may generally infeasible for users to

manually find and specify all the needed resources to meet the demands of jobs. Therefore, Resource Discovery requires a systematic way to express application requirements with respect to the resource information available in the grid Information System (GIS). A schema to describe the attributes of the systems in understanding the values mean for different systems, therefore, is required. This is an area of ongoing research with considerable argument about how to represent a schema (using LDAP, XML, SQL, etc.) and what structure should be inherent to the descriptions. Another important problem regards authorization filtering, which also needs a secure and scalable user accounting system. The availability of secure GIS publishing mechanisms would possibly allow publishing user to directly account for information in the GIS, from which this information should be automatically accessible to the scheduler [73], [75].

Resource Selection is the second phase in grid scheduling. This phase determines the best one from the list of suitable resources filtered from the first phase. This procedure of determination then needs the detailed dynamic information about the resources that are ranked on the basis of performance in this phase that is by facilitating the scheduler to choose the best ones out of all possible ones that ensure high performance in the execution of applications. Resource selection could be quite simple for sequential jobs, but very complex for parallel applications. In the second phase, the selection of the best match of jobs to resources is an NP-complete problem [18], [26], [29], [73], [74], [75].

Meanwhile, the Job Execution, the third phase of grid scheduling, could be very complex for various intermediary steps, like staging of files, advance reservation, etc. demanded from the preparation of a job run. In addition, due to the dynamic nature of grids, in which resource availability is changing constantly, a support of an automatic assignment of tasks to resources is necessary. Job execution and monitoring of the progress of job execution are two main activities of this phase. If job execution is not making adequate progress or is not meeting the required service level, then the grid scheduler may stop and reschedule the job execution. Such rescheduling becomes extensively harder for job executing in parallel on multiple sites. Dynamic scheduling algorithms are more demanding to perform such kinds of activities. Once the

allocation takes place; job monitoring starts by keeping the information about the execution progress as well as possible failures of jobs, and facilitating the grid scheduler to make the scheduling and rescheduling decisions accordingly [75], [74].

2.9 Types of scheduling in grids

There are number of types of scheduling exist in the grid system. Each scheduling type is based on some specific goals. An application might have different demands like task independent or dependent, batch or immediate mode etc. Each grid system has their own characteristics like centralized or decentralized approach, utilization of local schedulers, dynamics and so on. Grid scheduling has been divided into different types depending upon the needs of applications and grid characteristics[18], [29], [57]. The primary types of scheduling so far developed in the grid environments are as follows.

2.9.1 Independent scheduling

Scientific applications demand high computing power to process large amount of data. Computational grids has the capability to make parallel computation of various scientific applications. Each application or job is divided into different independent tasks. Scheduler makes allocation of tasks to various computational resources. Each task can execute independently at resource level. Computational grid has played a vital role in the efficient execution of various scientific applications like biomedical, digital animation, oil exploration, aviation, financial fields, etc. [18], [29], [57].

2.9.2 Grid workflows

Many scientific applications in grid have a lot of dependencies in their solution flow. Jobs or processes are dependent on one another. It is possible the output of one job/process might be the input for the next job/process. Each job can be divided into set of dependent tasks. The applications, which are composed of dependent tasks/jobs, are known as grid workflows [29]. These applications are very complex and make use

of grid computing in their controlled and efficient execution [78], [79]. Besides the efficiency, robustness is another major consideration in the grid workflows [18], [29], [57].

2.9.3 Centralized, hierarchical and decentralized scheduling

Grid scheduling is also divided from the management perspective like centralized, hierarchical and decentralized.

In centralized scheduling, grid scheduler keeps the information about all the resources and their state. It's very easy to control and manage the resources. But this type of scheduler is suffering from the scalability perspective and therefore is not suitable for large scale computational grid. Another issue with this scheduler, it might have single point of failure [18], [29], [57], [80], [81].

In hierarchical scheduling, schedulers can be organized in hierarchical way. The scheduler at the bottom in hierarchy keeps information about all grid resources and state of each resource. This scheduler has also the issues of scalability and fault tolerance. This scheduler is much better and more fault tolerant than centralized ones[18], [29], [57], [80], [81].

The autonomous grid sites make more challenging and complicated scheduling of application, job or tasks. There is not central control exists in decentralized or distributed scheduling. The local schedulers perform an important role in decentralized schedulers. The local users or other grid schedulers make request for scheduling of applications/jobs, which are delivered to local schedulers. Then local scheduler usually control and maintain the state of the job queue. Decentralized scheduling is more effective and practical for real grid environment but might be less effective than centralized one [18], [29], [57].

2.9.4 Static scheduling versus dynamic scheduling

Basically there are two important factors which highlight the dynamics of grid scheduling – (1) the dynamic of job execution, which indentifies the different

situations like failure of job execution, or stopping of job execution due to arrival of high priority job; and (2) The dynamics of resources, by which resources can join or leave the grid environment at any time. Numbers of resource are not fixed in grid system and vary over time. In addition, the local policies on using of resources could change with time; grid workload can significantly fluctuate over time and so forth. These two factors describe the behavior of the grid scheduler. If these factors are not present in grid scheduler, it means grid scheduling is static otherwise dynamic. As an illustration, in the static case, there is no job failure and resources are suspected available at all times (e.g. in Enterprise grids). Even though being unrealistic for many grids, it might be useful to take into account for batch mode scheduling: the number of jobs and resources is known as fixed during short intervals of time (time interval between two successive activations of the scheduler) and the computing capacity is deemed to be unchangeable. Other variants are possible to consider for example, just the dynamics of resources but not that of jobs [18], [29], [57].

2.9.5 Space-sharing and time-sharing approaches

Grid job scheduling policies can be generally divided into space-sharing and time-sharing approaches. Processors in time-sharing policies are temporally shared by jobs, while in space-sharing policies, are exclusively allocated to a particular job till its completion [82].

2.9.6 Immediate versus batch mode scheduling

These very well-known methods are mainly explored in distributed computing and also very helpful for grid scheduling. In immediate mode, jobs are planned once joining the system without waiting for the next time interval when the scheduler get activated or the job arrival rate is small, thus making resources available to execute jobs immediately. In batch mode, tasks are jointly grouped in batches and scheduled like a group. Batch scheduling conversely could acquire much better advantage of job and resource characteristics in determining which job to map to which resource since they dispose of the time interval between two successive activations of the scheduler [18], [57].

2.9.7 Adaptive scheduling

The overtime changeability of the grid computing environment demands adaptive scheduling techniques [83] that will well consider about both the current and future status of the resources with the goals of detecting and avoiding performance deterioration. Rescheduling considerably is a kind of adaptive scheduling in which running jobs are modeled to more suitable resources. [84] considered a type of grid applications with more and more independent tasks (Monte Carlo simulations, parameter-space searches, etc.), also known as task farming applications. For these applications with loosely coupled tasks, the authors designed a general adaptive scheduling algorithm. [85] stress the advantages of the grid system's capability to realize the state of the resources, and afterwards introduce a strategy for system adaptation in which grid jobs are managed, using an adaptable Resource Broker. [86] meanwhile described a scheduling algorithm developed on top of the GridWay framework, which utilizes the internally adaptive scheduling [18], [57].

2.9.8 Scheduling in data grids

Grid computing environments are making possible applications that actually work on distributed data as well as around various data centers. In such applications, it is crucial not only to assign tasks, jobs or application to the most effective and reliable nodes but also to reduce data movement and ensure a quick access to data. Simply, data location is significant in this type of scheduling. Actually, the effectiveness of the large computing capability of the grid might be compromised by slow data transmission that could be impacted by both network bandwidth and readily available storage resources. Thus, data generally should be ``close" to tasks to attain efficient access[18], [57].

2.10 Computational models for grid scheduling

How to define the grid scheduling problem then is becoming the main significant issue here. For this purpose, the most important and useful computational models are

presented in the following section, presenting their usefulness in the grid computing domain, particularly for resource allocation and scheduling [18], [57].

2.10.1 Expected time to compute model

In this model [87] the assumption is to dispose of estimation or prediction of the computational load of every task (e.g. in millions of instructions), the computing capability of every resource (e.g. in millions of instructions per second, MIPS), and an estimation of the past load of the resources. Furthermore, the Expected Time to Compute (ETC) matrix of size number of tasks by number of machines in which each location of *ETC[t][m]* indicates the expected time to compute task '*t*' at resource '*m*', is supposed to become known or computable in this model. In the simplest of situations, the entries *ETC[t][m]* might be computed by dividing the workload of task '*t*' by the computing capacity of resource '*m*'. This formulation is generally feasible for being possible to know the computing capability of resources while the computation demand for the tasks (task workload) could be known from requirements provided by the user from historical data or from predictions [18], [57], [88].

The ETC matrix model can describe various degrees of heterogeneity in a distributed computing environment through consistency of computing that identifies the coherence among execution times obtained by a machine with those acquired by the rest of the machines for a set of tasks. This feature is especially interesting for grid systems that purposively are to join in a single large virtual computer various resources, which range from laptops and PCs to clusters and supercomputers. Consequently, three types of consistency of computing environment, namely consistent, inconsistent and semi-consistent could be defined using the attributes of the ETC matrix [18], [57].

An ETC matrix is assumed to be consistent if each pair of machines is m_i and m_j, m_i executes a job more quickly than m_j, and m_i executes all the jobs faster than m_j. In comparison in an inconsistent ETC matrix, a machine m_i may execute some jobs faster than some other machine m_j and a few jobs slower than the same machine m_j.

Partially consistent ETC matrices are inconsistent matrices using a consistent submatrix of the predefined size [18], [57].

Furthermore, the ETC matrices are categorized based on the degree of job heterogeneity, machine heterogeneity and consistency of computing. Job heterogeneity means the degree of variance of execution times for all jobs in a given machine and machine heterogeneity means the variance of the execution times of all machines for the given job [18], [57].

From the explanation above, it could be observed that formalizing the problem instance is simple under the ETC model as it includes a vector of tasks workloads, a vector of computing capability of machines and the matrix ETC. It seems simple to define various optimization criteria within this model to evaluate the quality of a schedule. It is worth noting that incompatibilities amongst tasks and resources can be expressed in the ETC model; for instance, a value of $+\infty$ to *ETC[t][m]* would signify that task *'t'* is incompatible with resource *'m'*. Other restrictions of running a job on a machine could be simulated utilizing penalties to ETC values. It is, however, more difficult to simulate communication and data transmission costs [18], [57].

2.10.2 Total processor cycle consumption model

Despite its fascinating properties, the ETC model has an important constraint, specifically; the computing capacity of resources is assumed the same during task computation. This constraint becomes more apparent when it is considered that grid systems not only do the resources to have various computing capacities but also might change over time. The computing speed of resources may be assumed constant just for short or very short periods. In order to remedy this, [89] presented the Total Processor Cycle Consumption (TPCC) model - defined as the total number of instructions the grid resources that could finish from the beginning until the finishing time of execution of the schedule. As in the ETC model, the task workload is specified in number of instructions and the computing capability of resources in number of instructions computed per unit time. The total consumption of computing power due to grid application completion is measured. Obviously, this model

considers that resources could change their computing speed over time, as happens in large-scale computing systems, and as their workload is generally unpredictable [18], [57].

A problem instance in the TPCC model consists of the vector of task workloads [89] and a matrix expressing the computing speed of resources. Since the computing speed can change over time, one should fix a short time interval with the unchanged computing speed. Then a matrix PS (Processor Speed) is built over time in which one dimension is processor number and the other dimension is time; the component *PS[p][t]* represents the processor's speed during time interval *[t, t+1]*. As the availability and processing speed of a resource vary over time, the processor speed distribution is used. This model has shown to be useful for independent and coarse-grain task scheduling [18], [57].

2.10.3 Grid information system model

Though based on predictions, distributions or simulations, the computation models for grid scheduling introduced so far allow for a precise explanation of problem instance. Presently, other grid scheduling models are produced from a higher level perspective. In the grid Information System (GIS) model the grid scheduler utilizes task (job or application) file descriptions and resource file descriptions and also states information of resources (CPU utilization, number of running jobs per grid resource) provided by the GIS. The grid scheduler then computes the perfect matching of tasks to resources in line with the up-to-date workload information of resources. This model is not only much more practical for grid environments but also especially suited for the implementation of simple heuristics for example First- Come First- Served, Earliest Deadline First, Shortest Job First, etc. The problem instance in this model is developed at any point in time from the information on task file descriptions, resource file descriptions to the current state information on resources [18], [57].

2.10.4 Cluster and multi-cluster grids model

Cluster and multi-cluster grids refer to the grid model where the system consists of various clusters. The cluster grid of an enterprise for example consists of various clusters situated at various departments of the enterprise. One key goal of cluster grids is to provide a common computing infrastructure at enterprise or department levels by which computing services are distributed among different clusters. Clusters could belong to different enterprises and institutions; that is, to autonomous sites owning their local users (both local and grid jobs are run using resources) and usage policies [18], [57].

The most typical scheduling problem in this model is a grid scheduler, which uses local schedulers of the clusters. The benefit of cluster grids is to maximize the effective use of resources and, simultaneously, increase the throughput for user tasks. This model was used in [90] for scheduling data intensive bag-of-tasks applications. The problem instance in this model is built at any point in time from the information on task file descriptions; again, the assumption is how the workload of each task is known a priori. Alternatively, the multi-cluster grid could be formally represented as a set of clusters, each one with the information on its resources. Realizing that it is not necessary for the grid scheduler in this model to know neither the information on resources within a cluster nor the state information or control on each grid resource, it is possible to minimize dependencies on grid information services and follow local policies on resource utilization [18], [57].

2.11 Resource allocation and Job scheduling algorithms

The discussions in the previous sections reflect that grid scheduling obviously is challenging issue in which many constraints and optimization criteria have to be satisfied. A number of algorithms have been introduced for resource allocation and job scheduling.

2.11.1 Resource Allocation Approaches

Resource allocation is one of the phases in grid scheduling. Resources are geographically distributed between different time zones. Resource allocation is the mapping of jobs to available resources. A job is typically divided into tasks, which are allocated to different computers on a grid for execution. The actual execution time of a job is dependent on the method of resource allocation, and the number and sizes of the tasks. Hence, the resource allocation strategy plays a key role in grid scheduling [21], [91].

Types of resource allocation policies can differently be categorized into centralized and decentralized approaches. Traditional resource allocation uses a centralized approach. Jobs are assigned to the appropriate processors on the basis of the distribution strategy implemented by the resource scheduler. There are DAG (Directed a Cyclic) node weight based (i.e., task execution time) policies which allocate the resources to the jobs' tasks according to their pre-assigned weights [92], [93].

There are also cluster-based policies, which allocate the tightly coupled tasks to the same resource to decrease the communication cost. As being static in nature, all centralized policies cannot guarantee that the computing time of a job stays within an acceptable range as the number of jobs and processors increases. Centralized approaches are also not flexible enough to adapt to changes during the computing period [94], [95].

Whilst, decentralized resource allocation policies tend to integrate the job scheduling process and the job assignment process into a single process. These policies distribute the jobs among servers and processors dynamically. Such policies are widely used in grid computing. Software agents are widely used to implement decentralized resource allocation [96], [97], [98].

In [99], the authors propose a dynamic task allocation technique based on the "divide and conquer" principle and working in two phases. During the first phase, the network is mapped onto a hyper-grid which in turn, during the second phase, is successively divided up into hyper-grids of a smaller dimension. This second phase

works in a recursive fashion. The load-balanced hyper-grids of dimension *k* are divided into hyper-grids of dimension *k−1*. This division process continues until their dimensionality is equal to *'1'*. When this stage is reached, the tasks can be distributed amongst the nodes. The proposed resource allocation technique is dynamic, mixed, non-preemptive, adaptive and fully distributed. The contribution of this technique is its approach to the transfer and placement of decisions.

In [99], [100], UDA (User-Directed Assignment) is proposed for task allocation. This is the simplest task allocation strategy and maps each task in arbitrary fashion to the computing resources with the shortest expected starting time. The user does not need to know about status of that resource. In this case, the scheduler only helps to map the tasks to the resources. Task execution is very dependent on the resource itself - if the resource is lightly loaded or idle; the task may be executed immediately. The task otherwise will be executed whenever the resource becomes available again. Though this algorithm can be easily implemented, a critical disadvantage still emerges as the completion time is totally random [99], [100].

Fast Greedy, also called Minimum Completion Time, maps each task in arbitrary order to computing resources so as to deliver the shortest expected completion time without considering the minimum execution time. In this case, as this heuristic may result in the task to be executed for longer, it may cause the user larger costs [99] [100].

Three well-known batch mode scheduling algorithms, Min-Min, Max-Min and Sufferage, were proposed in [101]. They first created a list of tasks ready to be executed called the "task prioritizing" phase. In the second phase, the tasks in the list are scheduled to resources based on a heuristic; called the "resource selection" phase [101], [102].

Min-Min heuristic works in two phases. Firstly; in task prioritizing phase it constructs a list of tasks (i.e., pool) ready to be executed. The algorithm then computes the Estimated Completion Time (ECT) of each task for each suitable resource. The task with the minimum completion time is mapped to the specified computing resource. A resource that can execute a task with Minimum ECT (MCT) is

then chosen as the most suitable resource for that task. The task and resource are then paired. In the resource selection phase, the resource-task pair with lowest value of MCT is scheduled first. The corresponding task is deleted from the pool, and then the procedure will be repeated until no tasks remain. Min-min schedules the “best case” tasks first and generates relatively good schedules. Generally speaking, besides providing simplicity, rapidity and stability, the Min-min heuristic however possesses a drawback that is assigning the smallest task first and then a few larger tasks execute while several machines sit idle, resulting in poor machine utilization [99], [100], [101], [102].

Max-min heuristic is very similar to the Min-min algorithm. The only difference is that it schedules a resource-task pair with the highest value of MCT first [99], [100], [101], [102].

Min-Min and Max-Min heuristics are used with an expectation that tasks assigned to the machines can be in the earliest and fastest computation. In most of the resource allocation scenarios, Min-Min shows an outstanding performance [103], [104]. However, the study in [105] has proved that Max-Min can show a better performance than Min-Min when the lengths of tasks deviate greatly. For instance, with only one long task and many short tasks, Min-Min favors to execute all short tasks first, and the long task would be executed then while several machines sit idle. Max-Min in contrast executes the long task first. In the meantime, it executes short tasks concurrently with the long task. This can provide a better makespan and even a better resource utilization rate and load balancing than Min-Min.

Sufferage heuristic, an extension of Min-min algorithm uses the Sufferage value, which is the difference between the lowest and second lowest MCT. The pair of task and resource with the maximum Sufferage value is scheduled first. In practice, the algorithm gives priority to a task that would suffer the most if not being executed first [101], [106]. The Sufferage heuristic assumes that if the task is not mapped to this resource, the system will suffer the biggest loss. The higher the Sufferage value the task has, the higher its priority will be.

The drawback of Min-Min, Max-Min and Sufferage is that they do not consider the time required to transfer the required input files to the scheduled resource.

As an extension to Sufferage, [106] proposed the XSufferage algorithm for parameter sweep applications – currently has been widely used in the grid environment. The Sufferage value is computed not with one single computing resource, but with several different resources. The Sufferage value in this algorithm is computed taking into account the time required to transfer data file [106].

In [83] the author proposed an adaptive scheduling system by using a Max-min algorithm. The experimental results show that the proposed model can schedule tasks efficiently. The proposed system is particularly good at detecting and using idle processors. This system dynamically selects the proper scheduling strategy according to the accuracy of the predictor, considers the dynamic characteristics of grid applications and makes the scheduling adaptive to the grid environment.

[93] proposed the "Scheduling algorithm for heterogeneous processors with Different Capabilities" (SDC) based on the HEFT algorithm [107] to deal with tasks that can only be executed by certain resources. Tasks with "scarce capable resources" - the tasks that can be executed by few resources - are scheduled earlier so that they will obtain the required resources before other tasks that can be executed by many other resources [93]. However, since the ranking in HEFT based algorithms is based on the dependency of tasks [102], the tasks with "scarce capable resources" further down the workflow may still be blocked [93]. For example, at the beginning of a workflow, a scarce resource '*R*' might be allocated to an overlong task '*T*'. This can block the tasks further down the workflow that are independent of '*T*' but dependent on '*R*'. In addition, the definition of scarce resources in this work ("scarce capable resources" mentioned above) does not include the resources needed by several tasks. This justifies the need to include the notion of resource competition into the scheduling algorithm.

G. Murugesan proposed a resource scheduling model using DLT. The scheduling strategy divides the load equally into portions, each of which is allocated to a separate processor in such a way as to minimize the processing time. The author also

formulated an LP model for resource scheduling and conducted the experiment using the LINDO software package. However, a few shortcomings emerge with this resource scheduling model as it does not support the dynamic nature of the loads and resources. A random number of methods have been used for the division of a load, from multiple sources into equal amounts. The author also suggested that his work can be extended so that division of the load depends upon the resource capacity[108].

TORA is Windows based software and offers modules for solutions to different types of problems. It can be used for resource allocation problems and can be executed in both an automated and a tutorial mode. The automated mode results in the final solution of the problem, usually in the standard format. The tutorial mode is a unique feature that provides an instant feedback to test the reader's understanding of the computational details of each algorithm [109].

The following methods, relevant to resource allocation, can be used in TORA [109], [110].

- North West Corner Method (NWCM)
- Least Cost Method(LCM)
- Vogel’s Approximation Method(VAM)

However, TORA cannot be used to solve a computational problem that involves a large amount of jobs and processors. The procedural steps of NWCM and FCFS are similar. The LCM and Max-Min algorithms also have identical procedures for resource allocation.

2.11.2 Job Scheduling

Job scheduling plays a vital role in an efficient grid resource management. Most of the parallel jobs demand a fixed number of processors, which are unchangeable during execution [17]. Good job scheduling policies are very essential to manage grid systems more efficiently and productively [111].

In grid scheduling, a means of estimating the execution time of a task must be used and furthermore, information about capability and availability of each node must

be gathered. To match the tasks to nodes and monitor the tasks are necessary to do. The software to perform these management functions could be located either on a central computer, i.e., centralized, or on several computers, i.e., decentralized[112], [113].

Each node of a grid has its own local scheduling policy. When some nodes apply their priority policies in favor of local jobs, then global jobs making use of these nodes will suffer from the much longer response times, thus resulting in the overall performance of the grid to be degraded. In [114], the authors propose an adaptive site selection algorithm for a grid scheduler based on the priority policies of local schedulers. The experimental results show that the proposed algorithm can lower the difference with respect to average waiting times among the sites with different priority based scheduling policies. The proposed algorithm maintains a Remote Queue and a Local Queue at each node of the grid [114]. The drawback of this algorithm is too much processing time involved in accessing the large number of queues at the distributed nodes.

[115] proposes a compensation based scheduling approach to grid scheduling. This approach provides the predictable execution times by monitoring grid application performance, compares the monitored application performance with the desired application performance, and performs corrections by dynamically allocating additional resources. This approach has also been implemented and evaluated using the ALiCE grid system in which its scalability has been studied using a simulation. Experimental results then show that this approach is effective in reducing execution time estimation misses and the total execution times of grid applications. The authors also highlighted future work, which includes multi-resource compensation, resource partitioning and allocation, improvement in the execution time estimator, and the use of heuristics and dynamic methods for determining the value of a sensitivity factor in the application execution rate formula.

Grid job scheduling policies can generally be divided into space-sharing and time-sharing approaches. In time-sharing policies, processors are temporally shared by jobs. In space-sharing policies, conversely, processors are exclusively allocated to a single job until its completion. The well known space-sharing policies are First Come

First Served (FCFS), Backfilling, Job Rotate Scheduling Policy (JR), Multilevel Opportunistic Feedback (MOF), Shortest Job First (SJF), Shortest Remaining Time First (SRTF), Longest Job First (LJF), Priority (P) and Non Preemptive Priority (P-NP) approaches. The famous time-sharing scheduling policies on the other hand are Round Robin (RR) and Proportional Local Round Robin Scheduling [82], [116], [117]. For completeness, the most commonly used algorithms will now be explained.

The FCFS is the simplest and non preemptive CPU scheduling algorithm. For this algorithm the ready queue is maintained as a FIFO queue. Each new process is added to the tail of the ready queue and then the algorithm dispatches processes from the head of the ready queue for execution by the CPU. A process terminates and is deleted from the system after completing its task, it. The next process is then selected from the head of the ready queue [118], [119].

The SJF algorithm takes the processes using the shortest CPU time first. For this algorithm the ready queue is maintained in order of CPU burst length with the shortest burst length at the head of the queue. A new process submitted to the system is linked to the queue in accordance with its CPU burst length. The algorithm dispatches processes from the head of the ready queue for execution by the CPU. Similar with the one in FCFS, when a process has completed its task, it terminates and is deleted from the system. The next process is then dispatched from the head of the ready queue [119].

The SRTF algorithm is the preemptive flavor of the SJF algorithm. For this algorithm, the ready queue is maintained in order of CPU burst length with the shortest burst length at the head of the queue. When a new process is submitted to the system, the algorithm then checks if the new process requires less time than that remaining of the ‘active’ process, if so, then preemption occurs and becomes the new process’s turn for execution, if not, it is linked to the queue in accordance with its CPU burst length. The algorithm dispatches processes from the head of the ready queue for execution by the CPU. Again, when a process has completed its task, it terminates and is deleted from the system. The next process is then dispatched from the head of the ready queue [119].

The LJF algorithm takes the processes that use the longest CPU time first. For this algorithm the ready queue is maintained in order of CPU burst length with the longest burst length at the head of the queue. A new process submitted to the system is linked to the queue in accordance with its CPU burst length. The algorithm dispatches processes from the head of the ready queue for execution by the CPU. Having completed its task, a process terminates and is deleted from the system. The next process is then dispatched from the head of the ready queue [117], [119], [120].

Round-robin scheduling [50], [119] is a simple way of scheduling in which all processes form a circular array and the scheduler gives control to each process at a time. The ready queue for this algorithm is maintained as a FIFO queue. A process submitted to the system is linked to the tail of the queue. The algorithm dispatches processes from the head of the ready queue for execution by the CPU. Processes being executed are preempted on expiry of a time quantum, which is a system-defined variable. A preempted process is linked to the tail of the ready queue. When a process has completed its task, i.e., before the expiry of the time quantum, it terminates and is deleted from the system. The next process is then dispatched from the head of the ready queue. This algorithm produces a good response time as compared to other scheduling algorithms.

In the priority scheduling algorithm; the processes are prioritized in accordance with their operational significance. For this algorithm, the ready queue is maintained in the order of the system-defined priorities. Every process is assigned a priority and a new process submitted to the system is linked to the process in the ready queue having the same or a higher priority. The algorithm dispatches processes from the head of the ready queue for execution by the CPU. When a process has completed its task, it terminates and is deleted from the system. The next process afterward is dispatched from the head of the ready queue. If the priority criterion for execution is the order of arrival of the jobs into the system, then the priority scheduling behaves like FCFS scheduling. Alternatively, if the priority criterion is such that the jobs with shorter CPU burst lengths are assigned higher priorities, then this makes the priority scheduling behave like SJF scheduling [119].

[121] proposes a Self-Adjustment-Round-Robin (SARR) scheduling approach based on a dynamic-time-quantum algorithm. For this algorithm, the ready queue is managed as a FIFO queue. A process can execute up to the value of a computed time quantum for each round. This approach computes the time quantum, which is repeatedly adjusted according to the CPU burst time of the now-running processes and calculated by taking the median of the remaining CPU times of all processes in the ready queue. The minimum value for the time quantum used in the algorithm is 25 units.

In [122] the author proposed a Round Robin Priority Algorithm modified version of the round robin scheduling algorithm. This algorithm allows the user to assign priority to each process in the system and includes the concept of intelligent time slicing depending on two aspects - the process priority and the context switch. The time slice is computed for each process using range, process priority, total number of priorities and total number of processes in the ready queue. Range is computed by taking the average of the least and the longest CPU burst times in the ready queue and processes are assigned to the CPU for execution on the basis of priority and can execute up to the computed time slice for one time. The proposed algorithm has shown the good performance measures as compared to the round robin scheduling algorithm for different set of processes. The author also developed a web based simulation framework to study and evaluate the performance of various scheduling algorithms [122].

In [123], by the authors, a variant of the round robin scheduling algorithm was proposed. This simple round robin scheduling algorithm cannot be implemented in real time operating systems as it can cause too many context switches and results in a larger waiting time and turnaround time. This algorithm introduced the concept of an intelligent time slice - a combination of the original time slice, priority component, shortness component for CPU burst time and a context switch component. The proposed algorithm computes the time slice for each process manages the ready queue as the FIFO queue and executes each process for the computed intelligent time slice in a circular fashion.

A variant of the priority scheduling algorithm was also proposed in [124] . For this algorithm, the ready queue is maintained as a priority queue. Priority refers to some fixed ranges of numbers such as 0 to 9 (for 10 processes in the ready queue). The lowest number indicates the highest priority process and each process in the ready queue holds a priority number. The algorithm selects the highest priority process for execution on the CPU. If two or more processes hold the same priority, the process with the least CPU burst time will then be selected. While, if the processes have the same priority level and equal CPU burst time, then the proposed scheduling algorithm breaks the tie by using the FCFS scheduling algorithm. Experimental results shows that the proposed algorithm results in reduced average waiting time and average turnaround time as compared to existing priority scheduling algorithms [124].

The SRBRR (Shortest Remaining Burst Round Robin) scheduling algorithm is a variant of the RR scheduling algorithm proposed in [125]. The ready queue for this algorithm is maintained based on SJF. The SRBRR algorithm is compared with the RR scheduling algorithm using different case studies. The SRBRR algorithm produces better results in terms of reducing the number of context switches, average waiting time and average turnaround time in comparison to the RR algorithm. In [125], the dynamic time quantum technique has been used for scheduling of jobs. Time quantum is computed by taking the median of the remaining burst time of processes in the ready queue. SRBRR favors the shortest job for execution.

Several scheduling policies have been implemented in modern resource management systems for high performance computing. The first come first serve (FCFS) with backfilling [126], [127] is the most commonly used; as on average, a good utilization of the system and good response times of the jobs are achieved. However, with certain job characteristics, other scheduling policies might be superior to FCFS. For example, for mostly long running jobs, the longest job first (LJF) is beneficial, while the shortest job first (SJF) is used with mostly short jobs [128].

In [116], the authors have extended the working of basic space sharing techniques like FCFS, SJF and LJF, and proposed an SJF-backfilled scheduling heuristic. The main theme of this research was to backfill the shortest job first (length) to reduce the job killing probability. The proposed method also considers the reservation order of

jobs in making the scheduling decisions. In this way, the authors have achieved the advantages of both the backfilling and the SJF scheduling policies.

In [129], the authors proposed the idea of ‘backfilling’ - a space sharing policy that allows a scheduler to make better utilization of available resources by running jobs in a prioritized order. Smaller jobs are assigned a higher priority than larger queued jobs. It also requires that all job service times must be known before a scheduling decision is made. The proposed method has been evaluated using the IBM SP2 system.

In [130], the authors have performed an analysis of the processor scheduling algorithms using a simulation of a Grid computing environment. Three space-sharing scheduling algorithms (FCFS, SJF and P) have been considered for simulation.

[82] proposes Grid level resource scheduling with a Job Grouping strategy in order to maximize the resource utilization and minimize the processing time of jobs. The author has performed an experimental performance analysis of three space-sharing policies (FCFS, JR and MOF) and two time-sharing policies (Global Round Robin and Proportional Local Round Robin Scheduling). A combination of the Best Fit and RR scheduling policies is applied at the local level to achieve better performance. With RR, a fixed time quantum is given to each process, present in the circular queue, for fair distribution of jobs. It is also concluded that time-sharing scheduling policies perform better than space-sharing scheduling policies. The RR scheduling policy is extensively used for job scheduling in grid computing [82] [131].

In [132], the author introduced a dynamic scheduling model for parallel machines from an implementation perspective. The proposed model of a parallel job is based on a penalty factor. [132] also addresses open issues for the researchers. First, theoretical and experimental analysis of the idle regulation is needed with more variations of job scheduling strategies (largest job first, backfill etc.) and optimization criteria from both a user and a system perspective. Second, there is a need for an analysis of the system in dynamic scheduling environment that supports dependent jobs and jobs that can arrive at any moment.

2.12 Grid system performance and optimization criteria

Several performance requirements and optimization criteria have to be considered for grid scheduling. The grid scheduling is a multi-objective problem in its general formulation.

Grid performance criteria acquire account of CPU usage of grid resources, load balancing, system usage, queuing time, response time, cumulative throughput, waiting time and throughput, turnaround time. In fact, other criteria can be considered for characterizing a grid system's performance such as deadlines, missed deadlines, fairness, user priority, resource failure, and so on. Scheduling optimization criteria comprise of makespan, flow time, resource utilization, load balancing, matching proximity, turnaround time, total weighted completion time, lateness, weighted number of tardy jobs, weighted response time, etc. Both performance criteria and optimization criteria are suitable for any grid system; however, their success is dependent also on the considered model for example batch system, interactive system, and so on.

Two fundamental issues that have to be considered for the performance evaluation and comparison of grid scheduling algorithms are firstly, representative workload traces are required to produce dependable results, and secondly, a good testing environment should be set up, most commonly through simulations [133]. A standard workload should be used as a benchmark for evaluating scheduling algorithms [134], [135].

Grid Workload Archive played a key role in providing the grid workload traces for research purpose. Grid Workloads Archive (GWA) project is a platform for workload data exchange and a community center for the grid resource management and scheduling scientists. The GWA collects grid workloads (traces) from various contributors, and presents them to the public in a standard format (GWF). Theoretical analysis is difficult to apply for grid scenarios, and an extensive work has been done on different aspects in grid scheduling by using simulations[133]. Several simulations environments had been created to facilitate the evaluation. All of these simulation tools used discrete event-based simulations.

Grid scheduling algorithms cannot be designed without a good understanding of how today's grids are used, and of their performance. In addition, existing grid schedulers cannot be evaluated without understanding the characteristics of real grid workloads [136], [137], [138], [139]. Researchers have put a lot of efforts into real workload collection [140], [141], analysis[142], [143], and modelling [144], [145]. Workload characterization is important in order to understand the system performance. There is also a need for a tool to facilitate the researchers in performing the statistical analysis of the grid workload traces, and evaluating and improving the performance of grid schedulers. In this perspective, the study of the nature of real grid workloads is a vital step for improving the quality of existing grid schedulers.

Most of the resource allocation methods and job scheduling algorithms highlighted in the literature have not been evaluated using real workload traces. The aim of this thesis in turn is to propose new grid scheduling algorithms and evaluate their efficiency, performance and scalability in comparison to other well-known grid scheduling algorithms by simulation using real workload traces.

2.13 Chapter Summary

This chapter shows a survey of various computing architecture, grid systems and the grid scheduling problem. A number of computational models have been discussed for the grid scheduling problems. This chapter also classifies mechanisms developed and applied in numerous grid scheduling systems according to the taxonomy. Different types of scheduling based on different criteria, such as a static versus dynamic environment, centralized versus distributed etc. are identified. Various approaches for resource allocation and scheduling in grids are presented then. This chapter reveals the complexity of the scheduling problem in computational grids.

CHAPTER 3

GRID SCHEDULING MODEL

3.1 Chapter Overview

Scheduling is a fundamental issue in achieving high performance on computational grids. In this chapter, a grid scheduling model has been proposed. This chapter also presents the Linear Programming model for grid resource allocation. Besides proposing new algorithms for resource allocation and job scheduling, this chapter also presents the performance metrics, which will be considered for the evaluation of different scheduling algorithms. A performance evaluation strategy based on the proposed grid scheduling model is also included in this chapter.

The organization of this chapter is as follows: Section 3.2 presents the system of grid scheduling followed by Section 3.3 that presents the grid scheduling model and its components. Section 3.4 presents the new resource allocation method and Section 3.5 discusses the proposed job scheduling algorithms. The conclusion of the chapter at last is provided in Section 3.6.

3.2 Grid Scheduling

Grid systems are characterized by resource multiplicity and system transparency. Each grid system consists of a number of widely distributed and heterogeneous resources interconnected by a network. Besides providing communication facilities, grid enables resource sharing. A process may be executed remotely if the expected performance measure is better. From the user's perspective, the grid resources act like a virtual system. Therefore, when a user submits a process for execution, grid scheduler is responsible for controlling the assignment of resources to processes and

assigning the process to suitable computing node of the grid system according to allocation strategy.

Grid scheduler is the core component of a grid and is responsible for efficient and effective utilization of heterogeneous and distributed resources. Grid scheduling can be defined as a process of ordering tasks on computing resources and ordering communication between them, also known as the allocation of computation and communication over time [27].

A valid schedule is the assignment of tasks to specific time intervals of resources, such that no two tasks use same resources simultaneously, or the resource usage does not exceed the resources' capacities [146]. The schedule of tasks is optimal when minimizing a given optimality criterion (objective function). A scheduling problem is specified by a set of machines, a set of tasks, an optimality criterion, environmental specifications, and other possible constraints. The solution of a scheduling problem is an optimal schedule in the environment that satisfies all constraints. The functionality and performance of a grid scheduler much depends on the available features of the underlying local resource management systems. Now, this scheduling problem is shown to NP complete in nature [26], [147], means no deterministic algorithm exists which can solve this problem in polynomial time.

Therefore, these schedulers must be capable of collecting information describing the computational resources in grid as well as that of describing their current state and usage. Grid resource allocation and scheduling components are important for building computational grids, and also responsible for the selection and allocation of grid resources to current and future applications. Three main phases[26] of grid scheduling are following:

1. Resource Discovery
2. Resource Allocation
3. Job Execution.

The different phases as mentioned above are shown in Figure 3.1.

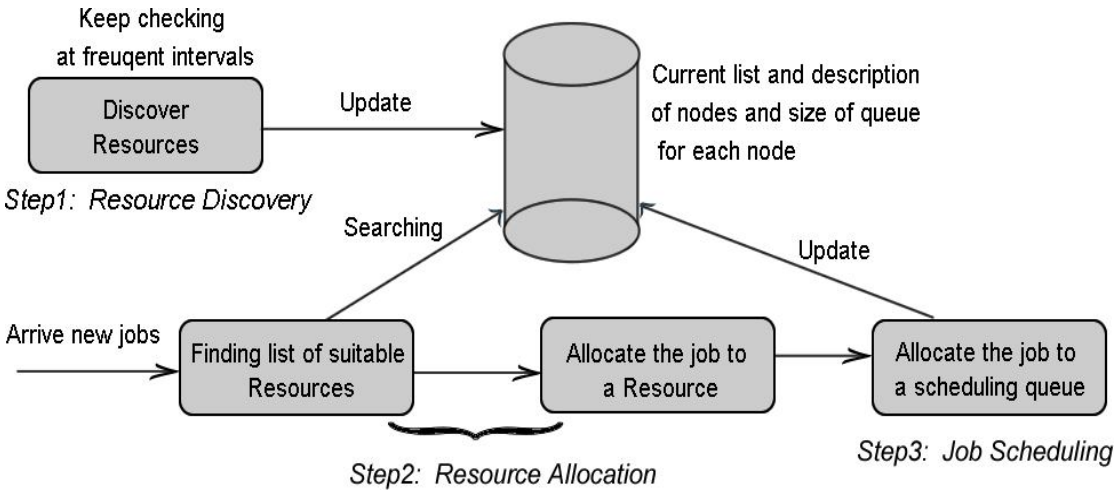

Figure 3.1: Phases of grid Scheduling

3.2.1 Resource Discovery

Being an important activity in grid scheduling, resource discovery in realistic grid applications is not feasible for users to find and specify the required resources at the time of jobs' composition. An efficient mechanism for this is needed by resource discovery to define the application requirements with respect to the resource information vector stored in the Grid Information System (GIS). Therefore, a schema to describe the attributes of the systems, in order to understand what the values mean for different systems, is necessarily to be proposed.

3.2.2 Resource Allocation

Phase two is resource allocation, involving a selection of feasible resources and mapping of jobs to resources. This selection procedure needs detailed dynamic information about the resources by accessing local resource description repository or querying performance systems. This information can be used to rank resources and allow the scheduler to choose the ones that should ensure high performance in the execution of applications. One the other hand, resource selection can be quite simple for sequential jobs, this task could become particularly complex for parallel applications. In the second phase, the selection of the optimal match of jobs to resources is an NP-complete problem [18], [26], [29], [73], [74], [75].

3.2.3 Job Execution

Third phase of this scheduling is job execution, which includes job execution at resource level. The Job Execution phase can be very complex since the preparation of a job execution can require various intermediary steps, such as, staging of files, advance reservation, etc. In addition, due to the dynamic nature of grids, in which resource availability can change constantly, a support of an automatic way to assign tasks to resources is necessary. One of the main activities of this phase regards the monitoring of the progress of an application execution. If a job execution does not make sufficient progress or meet the required service level, the scheduler may stop the job execution and then reschedule it. Such rescheduling is significantly harder for parallel job executing on multiple sites. Dynamic scheduling algorithms are required to perform this kind of operations. Once the allocation is complete, the monitoring system will inform about the execution progress as well as possible failures of jobs, depending on the scheduling policy that will be rescheduled or migrated to other resources [73], [74].

Grid scheduler uses several different algorithms at each phase for efficient and effective utilization of grid resources and to maximize the throughput of the grid. Proper grid scheduling can have a significant impact on the performance of the system. Thus, the aim of this thesis in turn is to provide new algorithms for effective resource allocation and job scheduling.

3.3 Proposed Grid Scheduling Model

To design new grid scheduling algorithms, a suitable scheduling model is required. Proposed grid scheduling model is an extension of the generic grid scheduling model [30], and it comprises of six components as shown in Figure 3.2. Further details on the six components are described in the following sub sections.

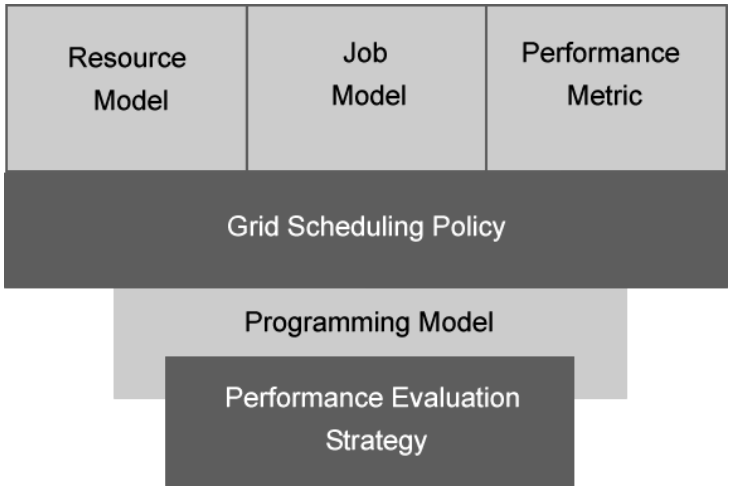

Figure 3.2: Grid scheduling model

3.3.1 Resource Model

Resource model describes the characteristics of resources. Different types of resource and their characteristics have already been discussed in chapter 2. In proposed grid scheduling model, only processor as resource has been considered and it was assumed that each processor forms a node of cluster or grid system. Therefore, the terms such as resource, node and processor are interchangeably used to have same meaning throughout in this thesis.

3.3.2 Job Model

Job model represents the characteristics of user jobs. In proposed grid scheduling model, each job can be divided into set of tasks that are assumed to be independent. Two type of jobs (i.e.; synthetic and real workload traces) have been used in scheduling. Each workload trace is set of independent jobs, tasks or applications. Each job is expressed as tuple of:

<Job ID, Submit time, Run Time, NumberofProcessors, User ID, Group ID…>

Job model is very important component of grid scheduling model that has strong significance in evaluating scheduling algorithms. Real jobs traces have been taken from the leading computational centre for the evaluation purposes and jobs have been analyzed and characterized in chapter 4.

3.3.3 Performance Metrics

Performance metrics are another important component of proposed grid scheduling model in choosing the criteria of optimality. In grid scheduling model, performance metrics include the average waiting time, average turnaround time, average response time, average bounded slowdown time, machine completion time and maximum job stretch time. The optimality of resource allocation and job scheduling algorithms can be achieved by minimizing the performance metrics measures. Each performance metric has been discussed in the following sub sections. Here, *'i'* denotes the process id and *'n'* specifies the total number of processors.

3.3.3.1 Waiting time

It is the time for which a process waits from its submission to completion in the local and global queues [119], [148]. Mathematically, it can be written as:

$$WaitingTime[i] = CompletionTime[i] - SubmissionTime[i] - RunTime[i]$$

Waiting time can be expressed in terms of *Turnaround time*:

$$WaitingTime[i] = TurnaroundTime[i] - RunTime[i]$$

Average waiting time can be written as:

$$AverageWaitingTime = \frac{\sum_{i=1}^{n} WaitingTime[i]}{n}$$

3.3.3.2 Turnaround time

The *Turnaround time* of the job is defined as the time difference between the *completion time* and *release time* [119], [148]. *Flow time* of the job is also known as the *Turnaround time* [146]

Mathematically, *Turnaround time* can be expressed as:

$$TurnaroundTime[i] = CompletionTime[i] - SubmissionTime[i]$$

Average turnaround time can be written as:

$$AverageTurnaroundTime = \frac{\sum_{i=1}^{n} TurnaroundTime[i]}{n}$$

3.3.3.3 Response time

It is the amount of time taken from when a process is submitted until the first response is produced [119], [148]. In interactive grid applications, response time is a very important parameter. Mathematically, it can be expressed as:

$$ResponseTime[i] = StartTime[i] - SubmissionTime[i]$$

Average response time can be written as:

$$AverageResponseTime = \frac{\sum_{i=1}^{n} ResponseTime[i]}{n}$$

3.3.3.4 Bounded Slowdown Time

Bounded slowdown [149] of a job can be expressed as

$$BSLD[i] = max(1, \frac{RunTim[i]+WaitingTim[i]}{max(RunTim[i], threshold)})$$

Turnaround time can be written as:

$$TurnaroundTime[i] = WaitingTime[i] + RunTime[i]$$

Bounded slowdown of a job can be expressed in terms of Turnaround time:

$$BSLD[i] = max(1, \frac{TurnaroundTim[i]}{max(RunTim[i], threshold)})$$

Although a *threshold* of *'10'* seconds has been used in many research works in the context of parallel jobs, scheduling to limit the impact of very short jobs on the average bounded slowdown. In this research, a threshold of *'60'* seconds will be used for experiment as recommended by [149]. It is due to the longer time possibly taken in grid scenarios jobs. *Average bounded slowdown* of finished jobs is a dynamic performance metric which can be expressed as:

$$AVG\ BSLD_{resource} = \frac{\sum_{i=1}^{s} BSLD(i)}{s}$$

where 's' is number of finished jobs

3.3.3.5 Machine Completion Time

Machine Completion time is defined as the time for which a machine 'm' will finalize the processing of the previously assigned tasks as well as of those already planned tasks for the machine. This parameter also measures the previous workload of a machine [18].

The $ready\_time[m]$ is the time when machine 'm' will finish the previous assigned tasks. Machine Completion time requires both ready time for a machine and expected time to complete the jobs/tasks assigned to the machine. This parameter is important to measure the processed workload so far for computing node. Mathematically, *Machine Completion Time* can be written as:

$$completion[m] = ready\_time[m] + \sum_{i \in jobs} RunTime[i]$$

3.3.3.6 Maximum Stretch Time of a job

Stretch of a job, also called slowdown of a job, is defined as the flow of a job over the processing time. In order to avoid the starvation situation from the grid system, it is also required to minimize the stretch of each job rather than the sum of stretches of all jobs. This motivates to compute another performance parameter, i.e. *Maximum Stretch time of job* (*Stretch$_{max}$*) [149], [150].

$$Stretch[i] = \frac{CompletionTime[i] - SubmissionTim[i]}{RunTim[i]}$$

$$Stretch_{max} = max\{Stretch[i] \forall i \in Jobs\}$$

3.3.4 Grid scheduling policy

Grid scheduling policy consists of following components:

- **Resource allocation algorithm**

  It is the way to find suitable resources for allocation and jobs' mapping, tasks or applications to processors.

- **Job scheduling algorithm**

  It is the way in which jobs, tasks or applications are being executed on processors.

Most of the scheduling approaches follow two levels of scheduling architecture [76] as shown in Figure 3.3.

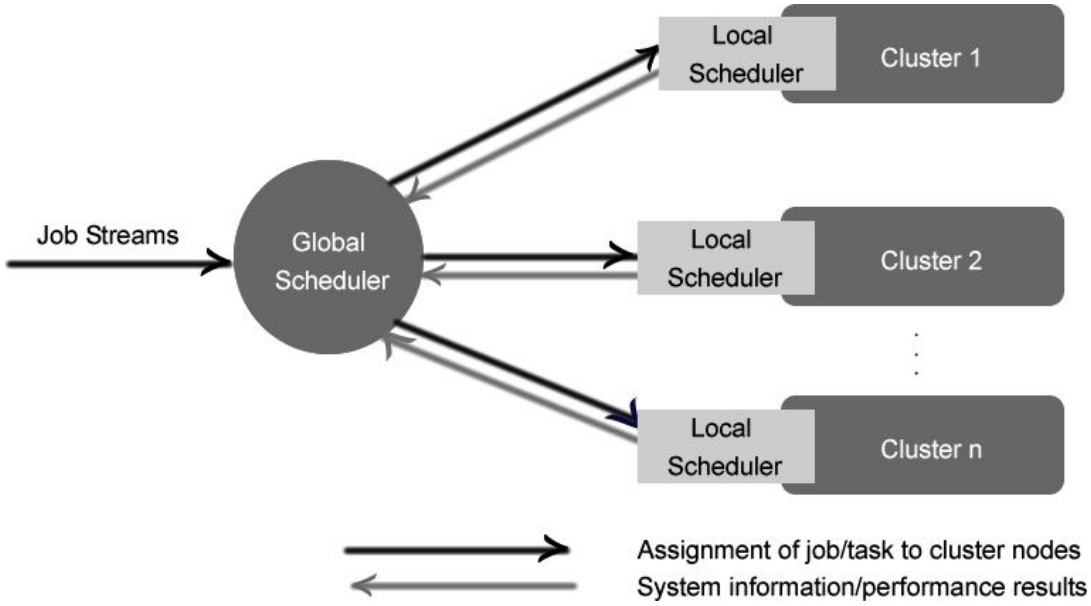

Figure 3.3: Grid Scheduling Architecture

In the proposed approach, the scheduling of jobs will be performed at two levels such as global level, i.e., grid level and local level.

- At the grid level, jobs are allocated to computing resources using resource allocation method. In order to improve the resource allocation strategy, a number of resource allocation methods have been evaluated using simulation

for a variety of grid resource allocation scenarios. This thesis has proposed a new method, called the Modified Least Cost Method (MLCM), for allocation of tasks to computing nodes aimed to minimize the computational cost in terms of time.

- At the local level, jobs are managed by the scheduler to execute the jobs or tasks on processors of the parallel and distributed machines. In this context, this thesis has proposed a number of scheduling algorithms, including Multilevel Hybrid Scheduling (MH), Multilevel Dual Queue Scheduling (MDQ), Dynamic Multilevel Hybrid Scheduling (i.e., MHM and MHR) and Dynamic Multilevel Dual Queue Scheduling (i.e., MDQM and MDQR). As scheduling is NP complete problem, a number of algorithms have been proposed to reach the optimality.

3.3.5 Programming Model

Programming model provides an environment to interact with the scheduler and describe detailed features of an application programming. Java and MPJ-express API (Application Programming Interface) have been selected for the development of scheduling simulators. The MPJ-express is widely used Java message passing library that allows writing and executing parallel applications for distributed and multi-core systems [151].

3.3.6 Performance Evaluation Strategy

Performance evaluation is the most critical step in testing and validating the efficiency and performance of scheduling algorithms. As per the standard practices [31], scheduling algorithms need to be evaluated using synthetic and real workload traces. Real workload traces have a strong impact in the performance evaluation of grid scheduling algorithms.

This thesis has presented the performance evaluation strategy, which has been designed using the standard practices [31], as shown in Figure 3.4.

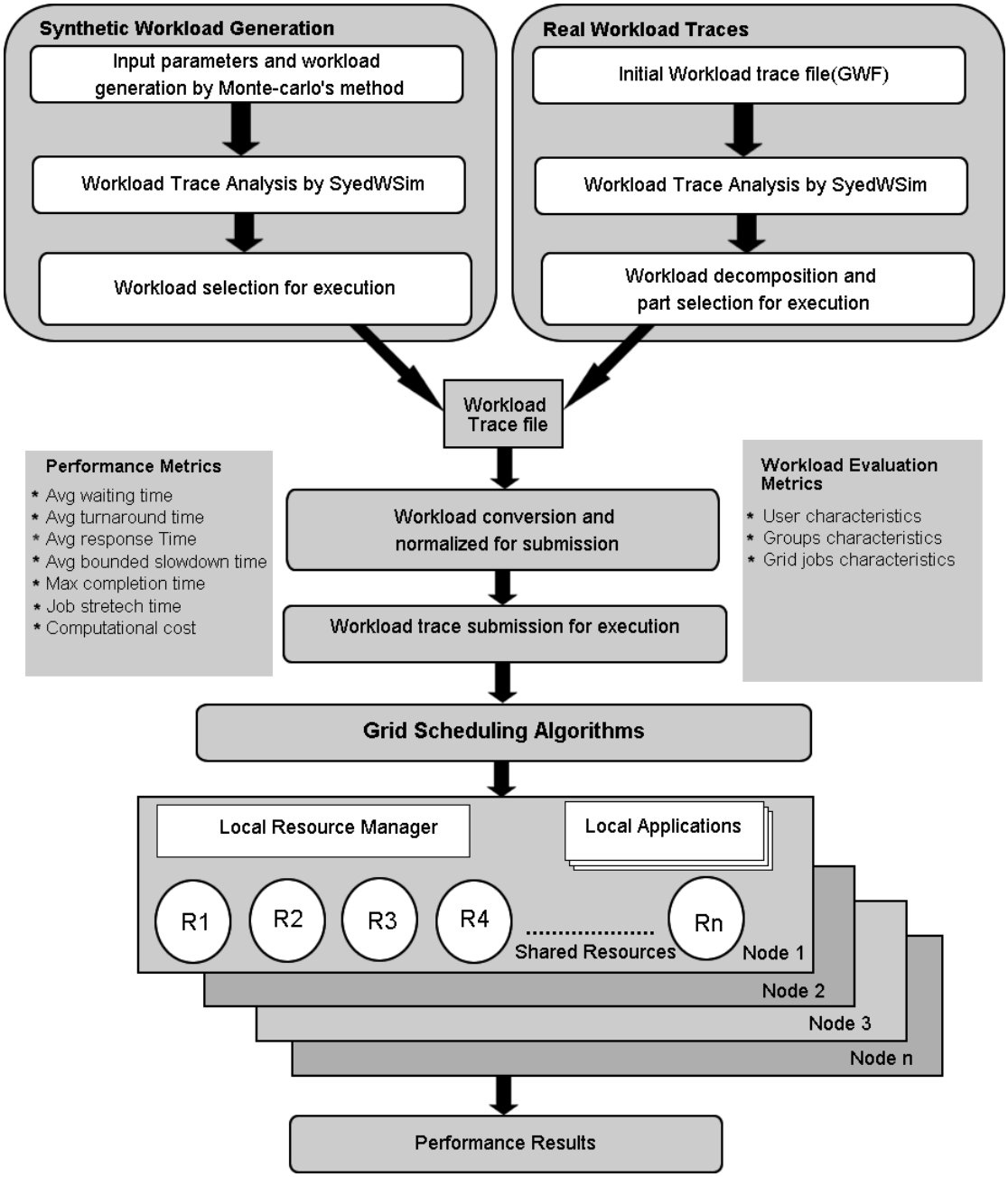

Figure 3.4: Performance Evaluation Strategy

The proposed performance evaluation strategy has been carried through the following activities:

1. Two types of workload (i.e.; synthetic and real workload traces) have been used in the grid scheduling evaluation. The synthetic workload traces have been produced using the Monte Carlo distribution method, meanwhile the real workload traces have been obtained from the grid workload archive [34].

2. ‘SyedWSim’, a new web-based simulator, has been presented in this thesis for analysis of grid workload traces. Detailed discussion on grid workload analysis is presented in Chapter 4.
3. The workload trace files have been produced by taking the workload segments from the whole workload traces and used as an input to the developed simulators for evaluation of resource allocation methods and job scheduling algorithms.
4. Proposed resource allocation method and other well known methods have been evaluated by simulation for the synthetic and real workload traces.
5. Proposed job scheduling algorithms and other well known algorithms have been evaluated using the synthetic and real workload traces on an experimental computational grid.
6. Results have been collected from the simulation and experimental grid.
7. Detailed comparative performance analysis of resource allocation methods has been carried out.
8. Detailed comparative performance analysis of job scheduling algorithms has been performed.

3.4 Proposed Grid Resource Allocation Method

Operation research is widely used in the grid scheduling models to achieve the optimum solution [18], [74], [152], [153]. In operation research, the transportation problem is a special type of Linear Programming problem. Transportation problem deals with the situation in which products are transported from a number of sources to a number of destinations. The objective is to minimize the total transportation cost of distributing all products from their sources to the destinations. The unit transportation cost is the cost of transporting one unit of the product from a source to a destination [110].

Grid resource allocation, a special case of Linear Programming (LP) transportation problem, is the issue of assigning tasks from a number of jobs to a number of processors at the minimum ‘allocation’ cost. The source and destination

correspond to job and processor respectively in the transportation problem. The following section presents a Linear programming model for grid resource allocation problem with the objective of minimizing the allocation cost. In this model, the cost is the total time to perform the following operations:

- Transfer in: Transfer of jobs to the resource
- Waiting Time: Time spent in the resource queue
- Computation Time: Actual time to execute the job
- Transfer out: Transfer of output files to the user.

3.4.1 Linear Programming Model

A grid is a computing system to process a number of jobs using a number of resources. In this study, the computational grid system is assumed to be composed of 'n' processors.

Let *J* be a set of *'m'* jobs and *P* be a set of *'n'* processors, as shown below:

$J = \{J_1, J_2, \ldots, J_m\}$

$P = \{P_1, P_2, \ldots, P_n\}$ where $n \geq 1$; $m \geq 1$

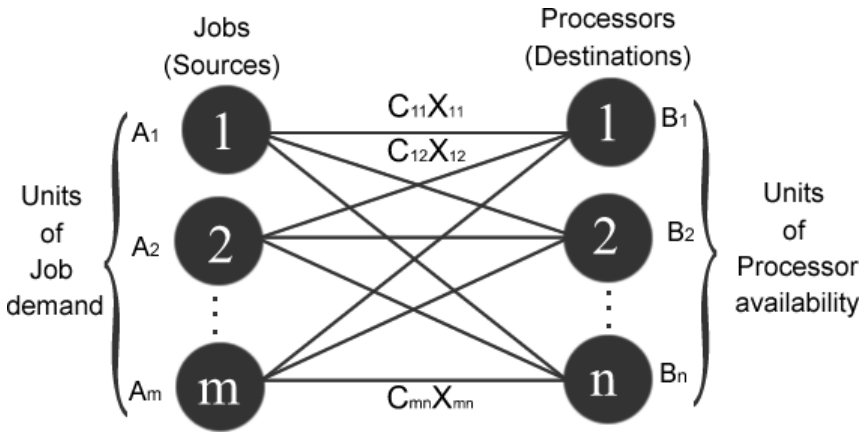

Figure 3.5: Resource Allocation Model

Figure 3.5 shows the mapping of jobs to processors. Each job is made up of a number of independent tasks, for which a job can be allocated to different processors. The variables used in the proposed resource allocation model can be defined as follows.

- Each job splits into *r* tasks. Mathematically it can be written as follows:
  $J_i = \{T_{i1}, T_{i2} \ldots T_{ir_i}\}$ where $r_i \geq 1$

- X_{ij} is the number of units of processing from job *i* to be executed on processor *j*.

- A_i is the number of units of processing making up job *i*.

- B_j is the number of units of processing that the processor *j* has been available.

- C_{ij} is the unit cost associated with allocating job *i* to processor *j*. This is made up of the transportation time and the time to execute one unit of processing. In what follows, C_{ij} will be referred as the 'allocation cost'.

- T_{ik} is the number of units of processing making up the k^{th} task of job *i*.

- X_{ijk} is a Boolean variable which is set to *'1'* if the k^{th} task of job *i* is executed by processor *j*. If this task is not executed by processor *j*, the variable is set to *'0'* then.

The main variables relevant to the grid resource allocation problem can be represented in Table 3.1.

Table 3.1: Representation of main variables

Jobs\Processors	P_1	P_2	……	P_n	**Size of Job (Workload) (*A*)**
J_1	C_{11}	C_{12}	……	C_{1n}	A_1
J_2	C_{21}	C_{22}		C_{2n}	A_2
:	:	:		:	:
J_m	C_{m1}	C_{m2}	……	C_{mn}	A_m
Processor Availability (B)	B_1	B_2	……	B_n	

The grid resource allocation problem is an LP problem. The objective function and the set of constraints are as follows:

Minimize

$$Z_{min} = \sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k=1}^{r_i} C_{ij} T_{ik} X_{ijk} \qquad (3.1)$$

Subject to

$$m > 0\,;\; n > 0\,;\; r_i > 0$$

$$C_{ij} \geq 0\,;\; T_{ik} \geq 0$$

$$X_{ijk} \in \{0,1\} \qquad (3.2)$$

$$\sum_{j=1}^{n}\sum_{k=1}^{r_i} T_{ik} X_{ijk} = A_i \quad ; \text{i=1, 2, 3… m} \qquad (3.3)$$

$$\sum_{i=1}^{m}\sum_{k=1}^{r_i} T_{ik} X_{ijk} \leq B_j \quad ; \text{j=1, 2, 3… n} \qquad (3.4)$$

Eq. (1) is the objective function (1) that is to minimize the computational cost (i.e, $Z_{\min}$). The minimization is subjected to a number of constraints stated in Eq. (2) – (4). The value of X_{ijk} in (2) indicates whether or not the k^{th} task of job i is executed by processor j. Constraint in Eq. (3) ensures that all the tasks for each job have been allocated.

Constraint in Eq. (4) ensures that the total quantity of processing units allocated to the job(s) must satisfy the processor availabilities.

3.4.2 Modified Least Cost Method (MLCM)

This section presents a new method (i.e.; MLCM) for allocating jobs to processors in grids to minimize the total allocation cost.

The Least Cost Method (LCM) is a well-known transportation method in Operations Research and used to find out the number of goods to be transported from each source to each destination in such a way that transportation cost will be minimized. LCM always matches a source with a destination using the least cost from

the transportation table. If the least cost is not unique then it chooses the source/destination combination that can transport the maximum amount of goods[110].

Proposed MLCM is a practical application of LCM with minor modifications for grid resource scheduling. In the resource allocation table, each job corresponds to a source and each processor corresponds to a destination. The cost for each job is given with respect to each available processor. MLCM introduces a new strategy for finding the least cost cell in the resource allocation table. Both LCM and MLCM favor the selection of the cell with the least allocation cost for mapping of a job to a processor. If the least cost is not unique then there is a tie. LCM breaks this tie by selecting the cell which can process the largest workload. MLCM however breaks the tie by selecting the cell which does not include the next least cost in its corresponding row or column. If a tie still occurs then MLCM breaks it by selecting the least cost cell which can process the smallest workload. MLCM not only makes maximum usage of the least cost cells for task-processor allocation to minimize the total allocation cost but also keeps in consideration all other next available least cost cells while making the scheduling decisions for further allocation. The detailed procedural steps of the MLCM are given in the Appendix A.

3.5 Proposed Job Scheduling Algorithms

Grid scheduling is an NP complete problem, i.e., no such deterministic algorithm exists which can generate an optimum result in polynomial time. Here, a number of approaches for efficient execution of user jobs in the grid environment have been presented - Multilevel Hybrid scheduling algorithms (MH) and Multilevel Dual Queue Scheduling algorithms (MDQ) have been proposed in this research work. These scheduling algorithms are based on a fixed time quantum value. To predict the demand of grid jobs in a dynamic scheduling environment however is not simple. The dynamic scheduling means jobs that are arriving in the system with different processing demands. However, algorithm based on the fixed time quantum cannot be a feasible solution to cater dynamic demands of processing for user jobs. To support the dynamicity of users jobs; two variants of MH - Dynamic Multilevel Hybrid

Scheduling Algorithm using Median (MHM) and Dynamic Multilevel Hybrid Scheduling Algorithm using square root (MHR) have been proposed in this thesis. This research work also proposed two more variants of MDQ, namely Dynamic Multilevel Dual Queue Scheduling Algorithm using Median (MDQM), Dynamic Multilevel Dual Queue Scheduling Algorithm using Square root (MDQR). For completeness, each proposed scheduling algorithm is explained as follows:

3.5.1 Multilevel Hybrid Scheduling Algorithm (MH)

Multilevel Hybrid scheduling algorithm (MH) uses master-slave architecture as shown in Figure 3.6. MH works in the two phases:

- **Phase 1:** MH uses the Round Robin(RR) allocation strategy for job distribution among the slave processors
- **Phase 2:** MH uses the proposed Hybrid Scheduling Algorithm (H) on each slave processor (i.e., computing node) for computation.

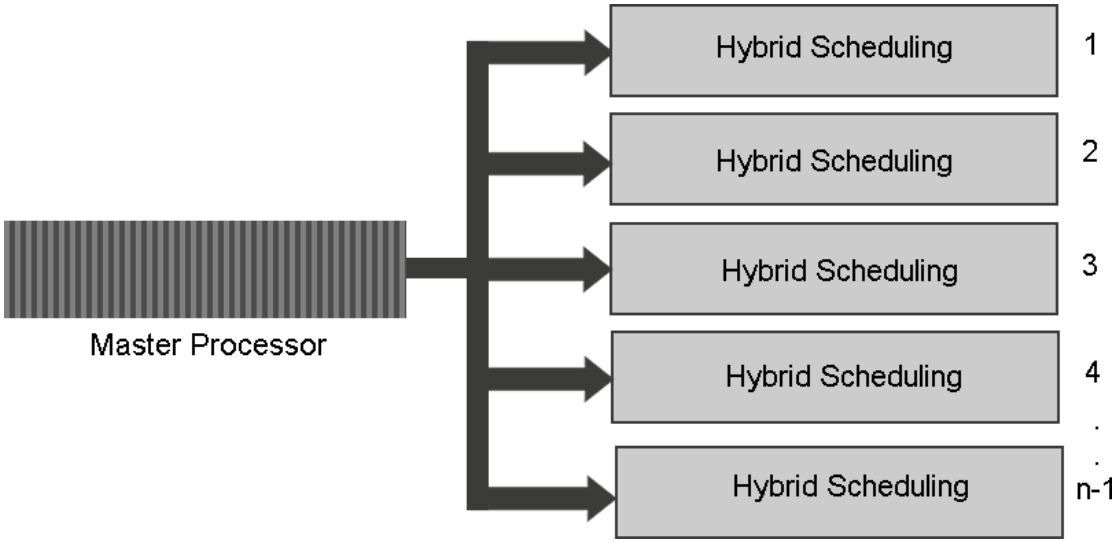

Figure 3.6: Block Diagram of MH

For H the ready queue is maintained in order of CPU burst length with the least burst length at the head of the queue. Two numbers are maintained. The first number, t_{large} , shows the burst length of the largest process in the ready queue, while the second one, t_{exec} , represents a running total of the execution time of all processes (since a reset was made). A new process submitted to the system is linked to the

queue in accordance with its CPU burst length. The process state diagram of H is shown in Figure 3.7.

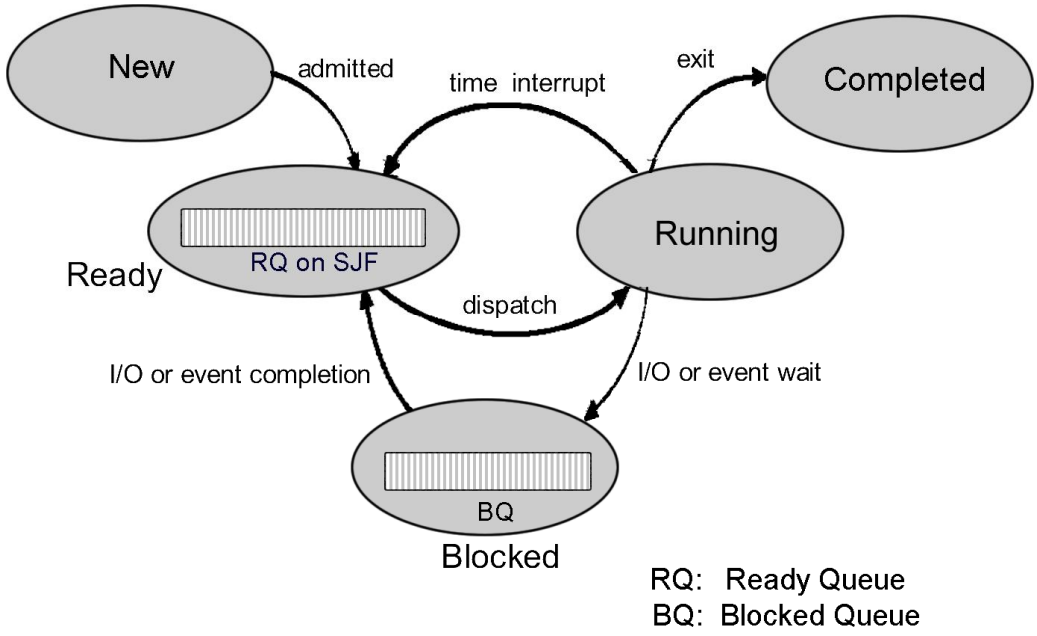

Figure 3.7: Process State Diagram of H

H dispatches processes from the head of the ready queue for execution by the CPU. Processes being executed are preempted on expiry of a time quantum, which is a system-defined variable. Following preemption t_{exec} is updated as follows:

$$t_{exec} = t_{exec} + quantum$$

The numbers are then compared.

If $t_{exec} < t_{large}$ then the preempted process is linked to the tail of the ready queue. The next process is dispatched from the head of the ready queue.

If $t_{exec} \geq t_{large}$ then the process with the largest CPU burst length is given a turn for execution. Upon preemption, the ready queue is sorted on the basis of SJF.

The value of t_{large} is reset to the burst length of the largest PCB, which is lying at the tail of the queue, and t_{exec} is reset to 0. The next process is dispatched from the head of the ready queue. When a process has completed its task; it is terminated and deleted from the system. t_{exec} is updated as follows:

$$t_{exec} = t_{exec} + time\ to\ complete$$

The numbers are then compared and the actions taken are the same as those for a preempted process.

The performance of MH scheduling algorithms is based on the value of a fixed time quantum. If the value of the time quantum is too small then MH results in too many context switches. If the value of the time quantum is too large then MH also loses its efficiency and behaves like the First Come First Served Scheduling Algorithm (FCFS). It means that in the MH it is necessary to know the nature and processing demands of jobs for setting the value of time quantum. In next following two sections, dynamic approaches have also been proposed to resolve the issues of the fixed time quantum encountered in MH. The proposed variants of the Multilevel Hybrid Scheduling Algorithm (namely Dynamic Multilevel Hybrid Scheduling Algorithm using Median (MHM) and Dynamic Multilevel Hybrid Scheduling Algorithm using Square root (MHR)) are as follows:

3.5.2 Dynamic Multilevel Hybrid Scheduling Algorithm using Median (MHM)

MHM is an extension of and works similarly as MH, yet, instead of fixed time quantum; it uses a dynamic time quantum approach. MHM computes the dynamic time quantum using the median of CPU times of processes in the ready queue. The dynamic time quantum approach has been taken from[121].

$$Time\ Quantum\ = median(\ C_1,\ C_2,\ C_3, \ldots .\ C_n)$$

where C_i is the CPU time of Process i and i ranges from *'1'* to *'n'*.

3.5.3 Dynamic Multilevel Hybrid Scheduling Algorithm using Square root (MHR)

MHR algorithm is another variant of MH that calculates the dynamic time quantum using square root of the average of CPU times of processes in the ready queue. MHR also computes the time quantum for each round and executes processes for the computed dynamic time quantum value. The objective of this approach is to reduce the number of context switches in the system.

$$Time\ Quantum\ = sqrt(avg(\ C_1,\ C_2,\ C_3, \ldots .\ C_n))$$

Where C_i is the CPU time of Process *i* and *i* ranges from *'1'* to *'n'*.

Proposed dynamic scheduling algorithms (MHM and MHR) will radically solve the fixed time quantum problem encountered in MH.

3.5.4 Multilevel Dual Queue Scheduling Algorithm (MDQ)

MDQ is based on a master/ slave architecture. A block diagram of MDQ is shown in Figure 3.8. MDQ also works in two phases:

- **Phase 1:** MDQ employs a Round Robin(RR) allocation strategy for job distribution among slave processors
- **Phase 2:** Dual Queue scheduling algorithm (DQ) is used on each slave processor for computation. Once a computation is done at the slave processor, then notification is sent to the master processor

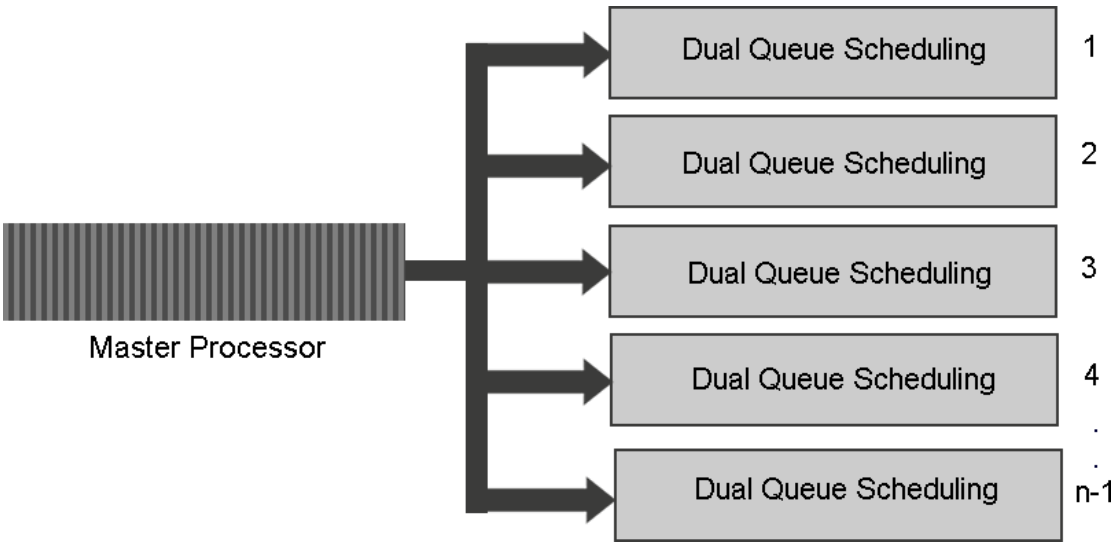

Figure 3.8: Block Diagram of MDQ

A process state diagram of DQ is shown in Figure 3.9.

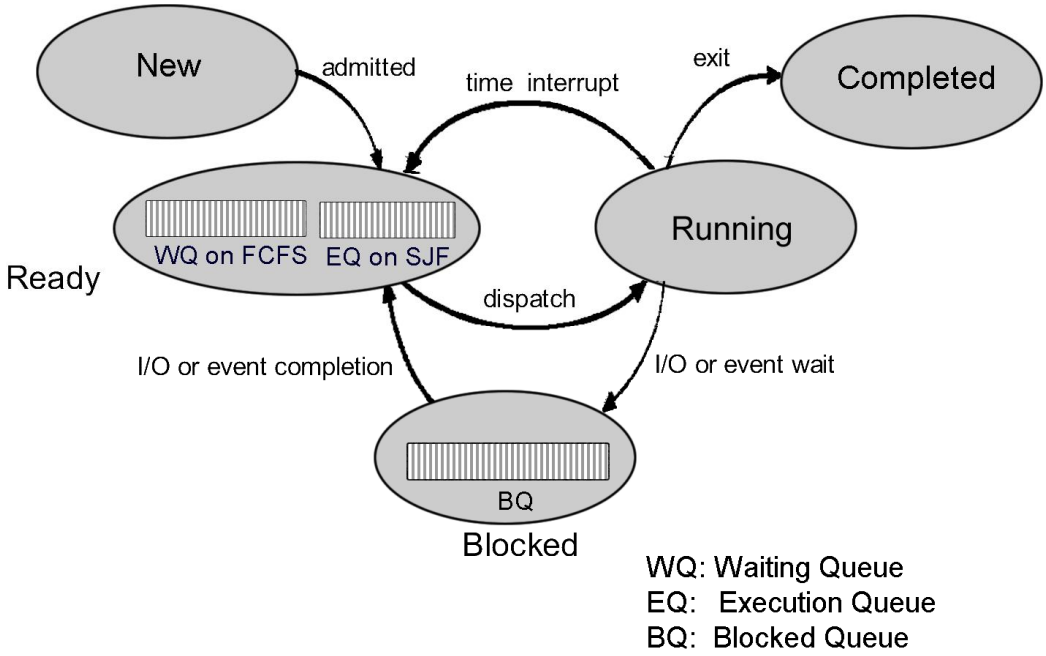

Figure 3.9: Process State Diagram of DQ

For the DQ algorithm, the ready queue comprises of two queues – the waiting queue and the execution queue. The waiting queue is maintained as an FIFO queue. A new process submitted to the slave is linked to the tail of the waiting queue. Whenever the execution queue is empty, all processes in the waiting queue are moved to the execution queue, leaving the waiting queue empty. The execution queue is maintained in order of CPU burst length, with the shortest burst length at the head of the queue. Two numbers are maintained. The first number, t_{large} , shows the burst length of the largest process in the ready queue (waiting queue and execution queue combined) while the second one, t_{exec} , represents a running total of the execution time of all processes (since a reset was made). The algorithm dispatches processes from the head of the execution queue for execution by the CPU. Processes being executed are preempted on expiry of a time quantum, which is a system-defined variable. Following preemption, texec is updated as follows:

$$t_{exec} = t_{exec} + time\ to\ complete$$

The numbers are then compared. If $t_{exec} < t_{large}$, then the preempted process is linked to the tail of the execution queue. The next process is dispatched from the head of the execution queue.

If $t_{exec} \geq t_{large}$, then the process with the largest CPU burst length is given a turn for execution. Upon preemption, all processes in the waiting queue are moved to the execution queue that leaves the waiting queue empty. The execution queue is then sorted on the basis of SJF.

The value of t_{large} is reset to the burst length of the largest process and t_{exec} is reset to *'0'*. The next process is dispatched from the head of the execution queue. Once a process has completed its task, it is terminated and deleted from the system. The following t_{exec} is then updated as follows:

$$t_{exec} = t_{exec} + time\ to\ complete$$

The performance of MDQ is also dependent on the value of a fixed time quantum. The much smaller the values of fixed quantum are, the much more context switches of MDQ will be. The MDQ loses its efficiency and behaves like the First Come First Served scheduling algorithm (FCFS) if the value is too large. The following sections present two more variants of MDQ (Dynamic Multilevel Dual Queue scheduling algorithm using Median (MDQM) and Dynamic Multilevel Dual Queue scheduling algorithm using Square Root (MDQR)). In these dynamic approaches, the value of time quantum value has been computed that depends on the processing demands of user jobs.

3.5.5 Dynamic Multilevel Dual Queue Scheduling Algorithm using Median (MDQM)

Proposed MDQM algorithm is a variant of MDQ. MDQM works as similarly as MDQ. However, MDQM uses a dynamic time quantum approach instead of fixed time quantum one and additionally computes the dynamic time quantum by taking the median of CPU times of processes in the ready queue. The dynamic time quantum approach is described in the details in [121].

$$Time\ Quantum = median(C_1, C_2, C_3, \ldots . C_n)$$

Where C_i is the estimated CPU time of process *i*, and *i* ranges from *'1'* to *'n'*.

3.5.6 Dynamic Multilevel Dual Queue Scheduling Algorithm using Square Root (MDQR)

Proposed MDQR algorithm is another variant of MDQ. MDQR computes the dynamic time quantum value using the square root of the average of CPU times of processes in the ready queue. MDQR also computes the time quantum for each round and executes processes for the computed dynamic time quantum value. This approach is aimed to reduce number of context switches in the system.

$$Time\ Qunantum\ = sqrt(avg(\ C_1,\ C_2,\ C_3, \ldots .\ C_n))$$

Where C_i is the estimated CPU time of process *i*, and *i* ranges from '1' to 'n'.

Proposed dynamic scheduling algorithms (MDQM and MDQR) aimed to solve the fixed time quantum problem encountered by MDQ.

The detailed procedural steps of the proposed job scheduling algorithms are given in the Appendix A.

3.6 Chapter Summary

This chapter presents a grid schedule model in which its components have been thoroughly explained. For grid resource allocation, here, the Linear Programming model been used in designing and developing the new resource allocation method is proposed as well. The new algorithms for grid resource allocation and job scheduling have been proposed in this chapter, in which for evaluation of grid job scheduling algorithms, some performance metrics are applied. This chapter has also presented the proposed performance evaluation strategy based on the standard practices for evaluation of grid scheduling algorithms. The strategy has been used for both analysis of grid resource allocation method and job scheduling algorithms. The details experimental results are provided in chapter 5 and chapter 6 respectively then.

CHAPTER 4

GRID WORKLOAD ANALYSIS

4.1 Chapter Overview

Grid computing is becoming the most demanding platform for solving large-scale scientific problems. Grid scheduling is the core component of a grid infrastructure. Grid scheduling plays a key role in the efficient and effective execution of grid jobs. In this context, understanding the characteristics of real grid workloads is a crucial step for improving the quality of an existing grid scheduler, and in guiding the design of new scheduling solutions. Towards this goal, this chapter presents a new web based simulator for the statistical analysis of grid workload traces. This web-based simulator provides a comprehensive characterization of the real workload traces. Metrics that have been characterized include system utilization, job arrival rate and inter-arrival time, job size (degree of parallelism), job runtime, data correlation and Fourier analysis. This simulator provides a realistic basis for experiments in resource management and evaluations of different job scheduling algorithms in grid computing.

The structure of this chapter is as follows: Section 4.2 presents the need of workload analysis tool. Section 4.3 describes the design and development of the web-based simulator. Section 4.4 is about the GUI of the web-based simulator. Section 4.5 describes practical applications of statistical theory. Section 4.6 is about the statistical analysis of real workload traces using developed simulator and section 4.7 concludes the chapter summary.

4.2 The need of workload analysis tool

New grid scheduling components cannot be designed without a good understanding of the working of existing grids. Also, existing grid schedulers cannot be accurately evaluated without understanding the characteristics of real grid workloads [137], [138], [139], [154] . In this perspective, the study of the nature of real grid workloads is a vital step for improving the quality of existing grid schedulers.

Different researchers use different programs for analyzing the workload traces. A number of approaches have been developed for statistical analysis of workload traces. In this work, an alternative web based approach to simulate such environments that uses resource utilization traces from real deployments has been developed. This simulator works on the statistical analysis of grid workload traces. A detailed analysis of any real grid workload trace can be obtained to quantify the performance of the grid systems from different perspectives, e.g. users, groups and individual jobs characteristics. This simulator takes the real trace as input in the grid workload format (GWF) [34], [155] and produces a variety of graphs to analyze the characteristic of workloads. SyedWSim also been used to analyze two well-known traces from two real scientific grid environments [34], [155], namely LCG1 and AuverGrid.

In the following sections, proposed simulation technique and results from developed SyedWSim have been discussed. The results from the SyedWSim have been compared and verified with the results available [34], [155].

4.3 Framework of Web based simulator (SyedWSim)

This section presents the framework for implementing a web based system, SyedWSim, for statistical analysis of workload traces. The goal of this framework is to visualize the performance statistics of grid from various perspectives and also provides the realistic basis for evaluation of grid scheduling algorithms. Framework of SyedWSim is shown in Figure 4.1.

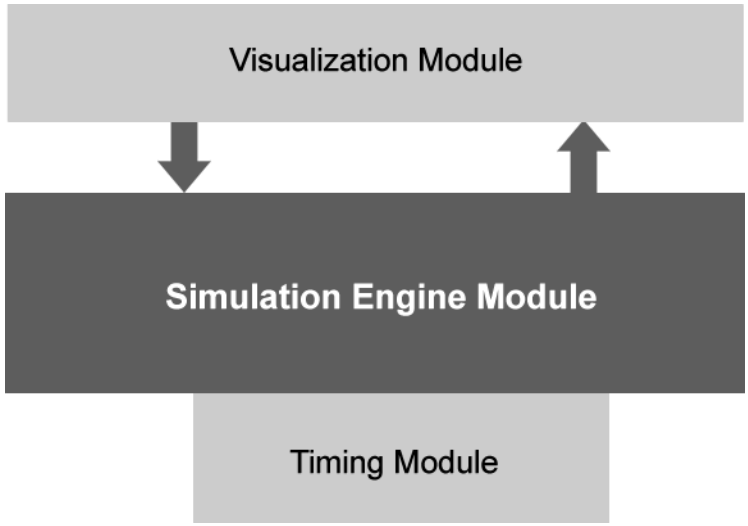

Figure 4.1: Framework of SyedWSim

SyedWSim frame work consists of following modules

1. Visualization module
2. Timing module
3. Simulation engine module

4.3.1 Visualization module

Visualization module provides a web-based interface to the user. It takes real workload trace (also called resource utilization trace) in the GWF format [34], workload percentage and interval size as input. It provides a number of options to users to choose to perform statistical analysis of workload trace from user, groups and jobs perspectives. Visualization module sends the user inputs to the simulation engine module for analysis purposes.

On receiving the computed statistical performance measures from the simulation engine, visualization module displays them graphically in an interactive way. This module has provided the user a number of features, like zooming, color palette etc, to visualize the statistical measures in a friendly way. Tooltip shows the relevant statistics at the mouse cursor position for each diagram, for each available option in SyedWSim.

4.3.2 Timing module

Timing module depicts the system's dynamic behaviour. It works with the Simulation engine module and makes use of the jobs' time units, as given in the workload file. It uses the trace files at runtime to generate data depicting how the model dynamics are changing under timing constrains. These performance measures can also be visualized and evaluated by visualization module.

4.3.3 Simulation engine module

Simulation engine module is the core module of the SyedWSim. It models the impact of different scheduling policies on grid performance under various loads and policies. It converts a source workload file into a relational data structure at runtime for analysis purposes. It uses the input workload trace and timing model, and characterizes the workload from user's perspective, group's perspective and grid jobs perspective. It makes use of a number of functions to characterize the workload using different statistical measures e.g., auto correlation function, Fourier transformation etc.

Workflow diagram of SyedWSim is shown in Figure 4.2.

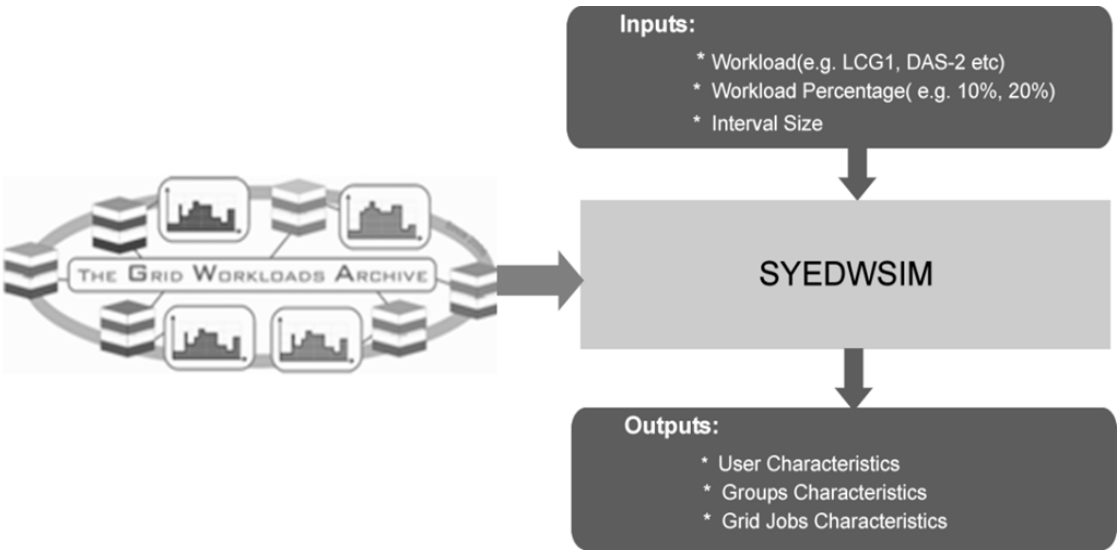

Figure 4.2: Work flow diagram of SyedWSim

The core functionality of the SyedWSim simulator is as follows: The main simulator loop operates on a timer that records time in the simulated system. When reading the input trace file, the simulator uses this timer to access workload data at the corresponding time stamp associated with it. Finally, SyedWSim models to the users' characteristics, groups' characteristics and grid jobs' characteristics.

4.4 Design and Development of SyedWSim

'SyedWSIM' has been designed and developed using Java. The class diagram of SyedWSim is shown in Figure 4.3.

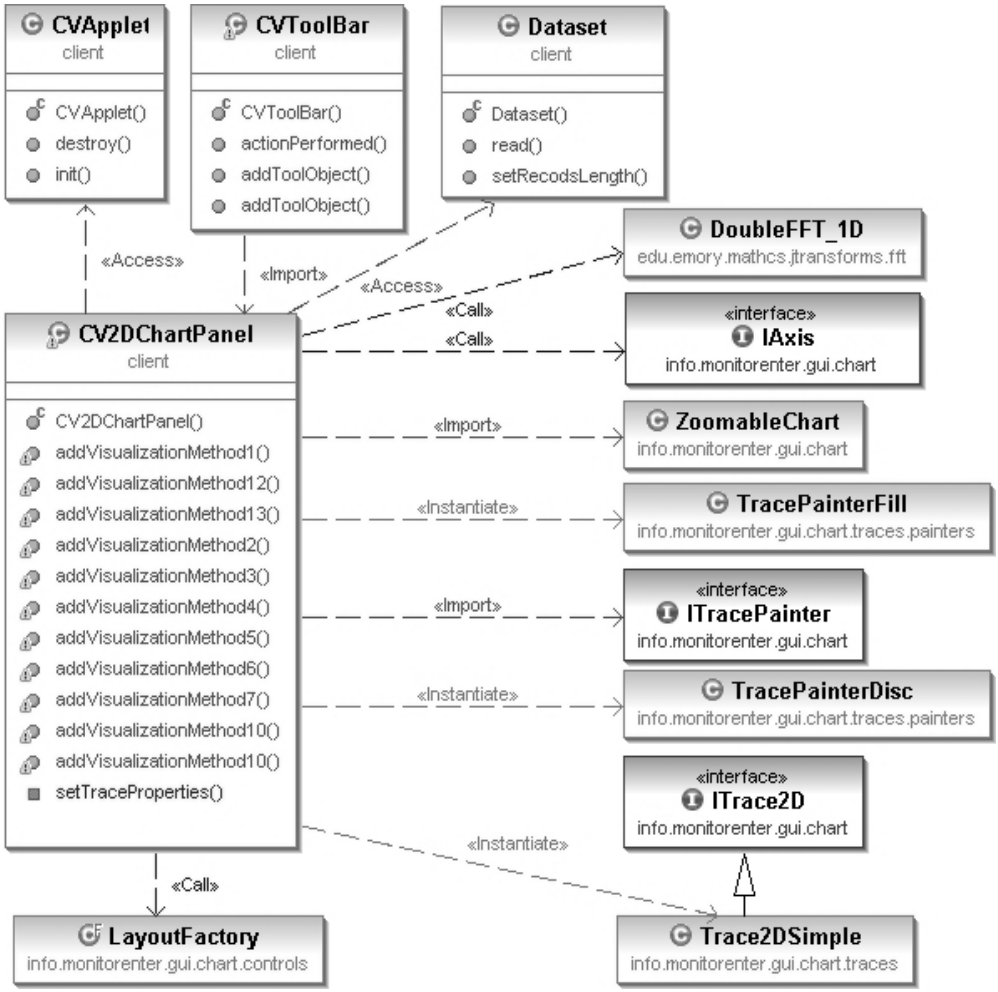

Figure 4.3: Class Diagram of SyedWSim

Class diagram has shown the object oriented design of SyedWSim. The descriptions of its components are as follows:

4.4.1 CV2DChartPanel

This class includes graph drawing functionality for all features of the SyedWSim related to the grid users, virtual organizations and jobs. It interacts with all other classes of the SyedWSim and produces output depicting the nature of the workload under analysis. This class includes *13* methods (namely AddVisualizationMethod1, AddVisualizationMethod2, etc.) as shown in Figure 4.3, and each method corresponds to each feature available with SyedWSim (i.e. user & total jobs, user & log(total jobs), etc.) as highlighted by 'Options' in Figure 4.4.

4.4.2 CVApplet

It is a web based applet file, which executes on a client browser. It includes a visualization class that extends the functionality of the Java Applet class. It also includes all visual components and provides all visual interfaces to interact with the SyedWSim.

4.4.3 CVToolBar

This class provides a toolbar, which includes a list of options allowing the user to experiment with the SyedWSim for workload analysis.

4.4.4 Dataset

This class file is used to download the dataset from the original text file, which is in GWF format, and to construct the dynamic data structure at runtime.

Open source codes are also being used in the development of SyedWSim. The 'JTransforms' package [156] has been used to apply the Fast Fourier Transform (FFT) to the values obtained by the autocorrelation function. A 'GUI' package, entitled JChart2D [157], is also used to produce the 2D Chart. The JChart2D is used for displaying the data contained in an ITrace2D. JChart2D inherits a number of features from javax.swing.JPanel. The package has facilitated for displaying the variety of

graphs. Another open source math package, entitled Commons Math [158], is also used for mathematical and statistical calculations.

4.5 GUI of SyedWSim

SyedWSim is a web-based application which facilitates this research for analysis of grid workload traces and also provides the realistic basis for evaluation of grid scheduling algorithms. SyedWSim provides a user-friendly way to perform large-scale simulations of multi-grid environments. The snapshot of graphical user interface of SyedWSim is shown in Figure 4.4.

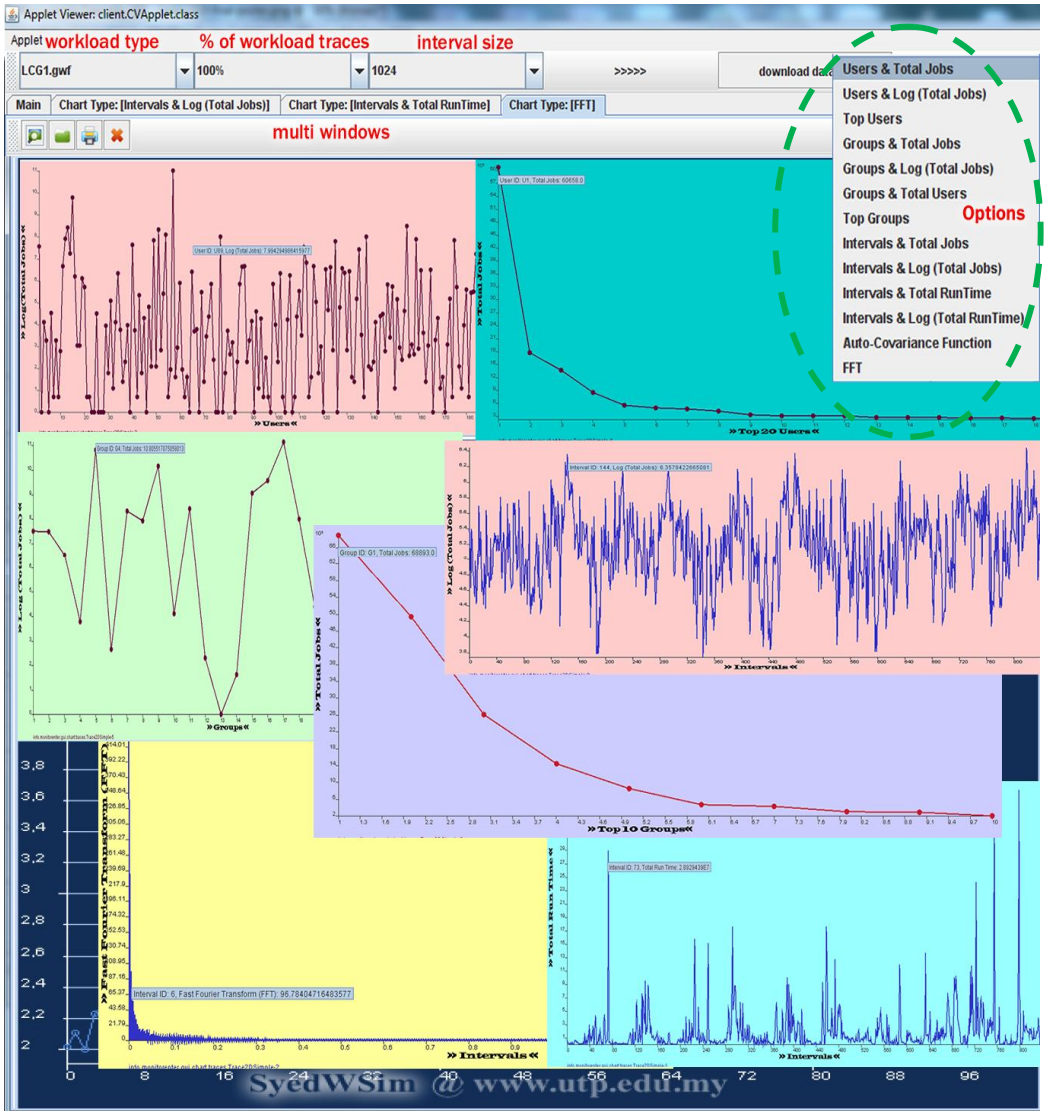

Figure 4.4: GUI of SyedWSim

4.6 Practical application of statistical theory

Two statistical analysis techniques have been implemented in SyedWSim. One implemented technique is autocorrelation whilst the second one is Fourier analysis. Autocorrelation is the cross-correlation of data with itself. It is a function to find the similarity between a list of observations, and the same list offset by a certain 'lag.' It is a mathematical tool to find repeating patterns and to study the correlation structure of single process. Fourier transforming the autocorrelation function (ACF) yield the power spectrum [133], [159].

Fast Fourier transformation (FFT) is a systematic way to perform Fourier transformation in short time on large amount of data. From the theory of the theory of Fourier analysis, it is known that periodicity shows up as peaks in the frequency domain. Real world data, however, seldom exhibits perfectly periodic behavior. In most situations, pseudo-periodic signals are observed instead, potentially arising from various sources of noises and time varying nature of generation schemes. From this perspective, it is necessary to use quantitative methods to measure the degree of periodicity in the data. Periodicity in a process can be detected and quantified using power spectrum based methods [133], [159].

Besides studying how events of the same process are correlated with each other, it is also important to reveal the correlation between events of distinct random processes. The simplest way of investigating is to plot samples of both variables and visually identify if any pattern exists. Self similar and long range dependent (LRD) processes are two important classes of generally scaling processes and LRD is highly relevant in context of this research work. In network traffic, both inter arrival and count based measure to be useful in analyzing the scaling behavior [133], [159].

From performance evaluation perspective, it is also desirable to include users and groups (virtual organizations) in the grid workload analysis since most of the policy rules are based on their names[133], [159].

4.7 Statistical analysis of workloads using SyedWSim

Grid workload archive [34] is an example repository for grid workload traces. In [159], a comprehensive statistical analysis has been carried out for a variety of workload traces on clusters and grids.

The workload analysis focuses on three aspects: user characteristics, group characteristics and system-wide characteristics (e.g., system utilization, job arrival rate, job characteristics). SyedWSim facilitates in a quick analysis of grid workload traces for the expert user and provides a detailed view of one grid.

This research work has reproduced the graphs of [159] to study the behavior of the dynamic nature of workloads 'LCG1' and 'AuverGrid'[34] , using 'SyedWSim'. Details about the format of workload traces LCG1 and AuverGrid are shown in appendix *'B'*. The number of jobs arriving in a particular period is its 'job count'. In the following analysis, trace job entries that have a negative runtime or a negative number of allocated processors have been dropped. The input GWF format of the LCG1 trace is shown in Table 4.1.

Table 4.1: Trace LCG1

1	2	4	5	12	13	16	17	18
Job ID	**Submit time**	**Run Time**	**NProcs**	**User ID**	**Group ID**	**Partition ID**	**Orig Site ID**	**Last Run Site ID**
1	1132444805	83	1	U1	G1	1	SWF	SWF
2	1132444808	3611	1	U2	G2	2	SWF	SWF
3 …	1132444817	205	1	U1	G1	3	SWF	SWF

The format of the trace is described as follows. Description of columns *1, 2, 4, 5, 12* and *13* correspond to *Job Id, Submit time, Run time, User Id* and *Group Id* respectively. Some columns of the trace are filled with *'-1'*. This means that the data are not given. Only the first three of the *188,041* jobs are shown in Table 4.1. One or more jobs were submitted by a user. There are different groups of users. As an

example, the first job in the trace means user *U1*, who is a member of group *G1*, submitted job *1*. The submit time of each job is given in seconds, and this is in chronological order. The time of the first job is *1132444805*. The time between a job's submit time and the submit time of the succeeding job is referred to as the 'inter-arrival time.' For example, job *1* was submitted at time *1132444805* whilst job *3*, which was the next job for the G1, was submitted at time *1132444817*. The gap between this pair of jobs, i.e. the inter-arrival time, is *12*.

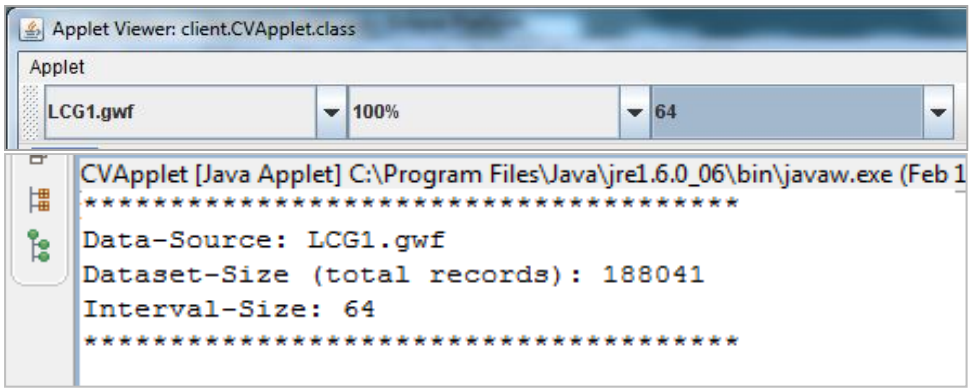

Figure 4.5 (a): The user jobs for LCG1

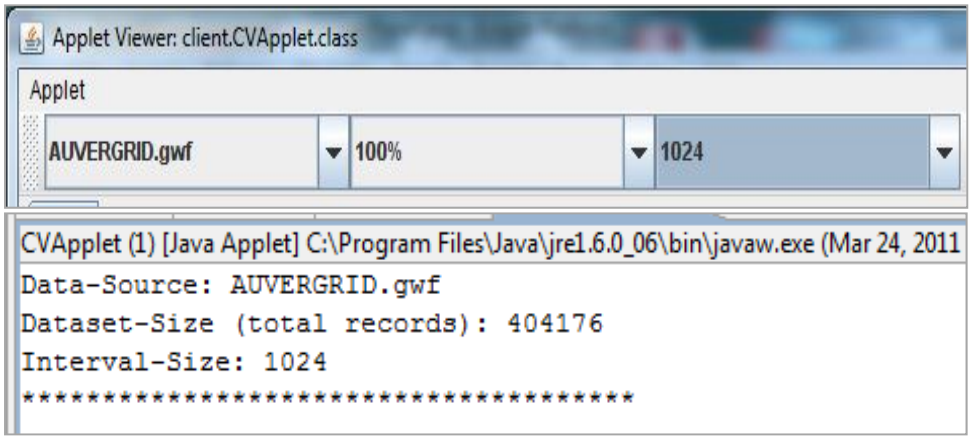

Figure 4.5 (b): The user jobs for AuverGrid

Figure 4.5(a, b) shows the user input for the workload traces LCG1 and AuverGrid respectively. The 100% workload has been used as input for processing and analysis purpose using SyedWSim. The total numbers of jobs in LCG1 and AuverGrid are *188041* and *404176*, respectively. The set interval sizes for LCG1 and AuverGrid are *'64'* and *'1024'* seconds respectively. The number of jobs arriving in each interval, referred to as the 'job counts'.

4.7.1 Users Characteristics

Firstly, this research work has analyzed the workload traces from the user's perspective using SyedWSim. Following figures have shown the users jobs and top users for LCG1 and AuverGrid workloads.

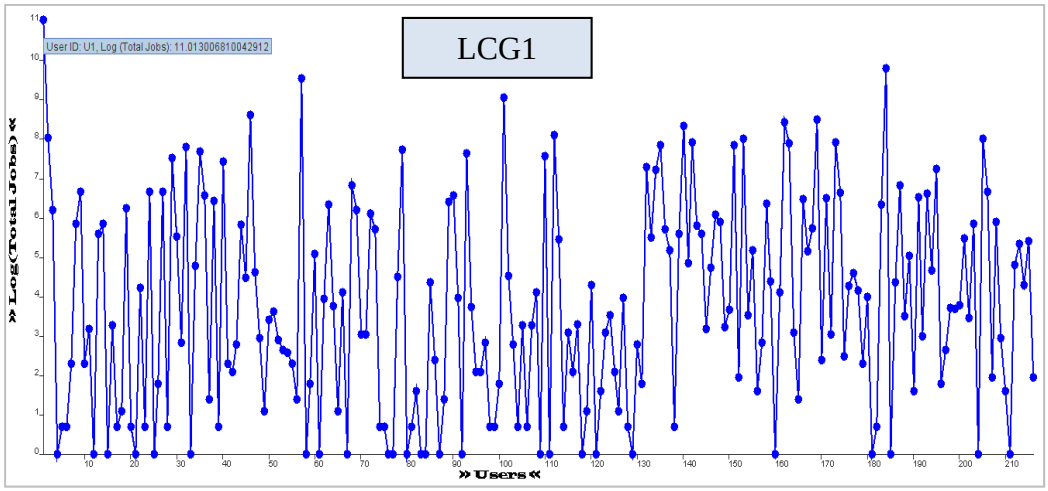

Figure 4.6(a): The user jobs for LCG1

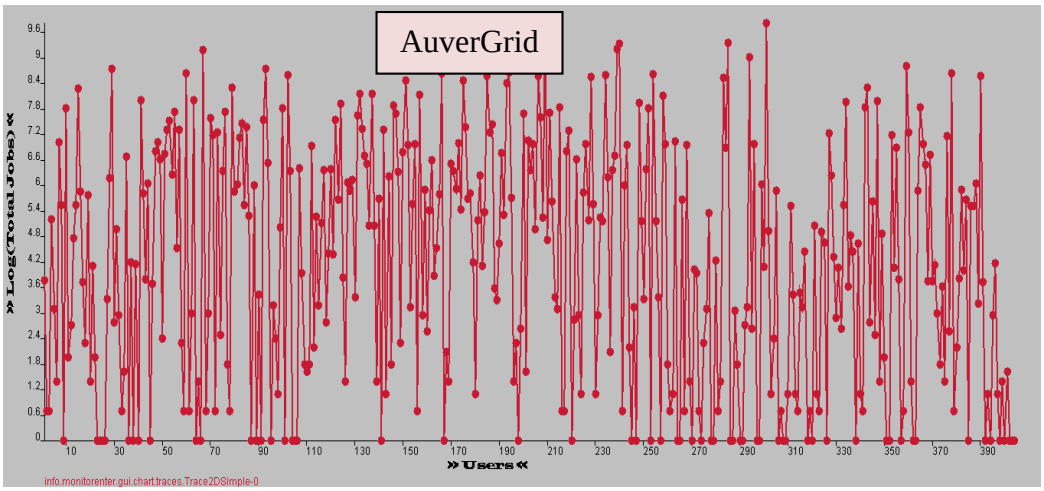

Figure 4.6(b): The user jobs for AuverGrid

Figure 4.6(a, b) shows the number of jobs per user for LCG1 and AuverGrid. The jobs submitted by some users are of a different order to those submitted by others. For LCG1, Figure 4.6(a) shows that user *'U1'* has submitted *60658* jobs, while user '*U2'* has submitted *3305* jobs; and so on.

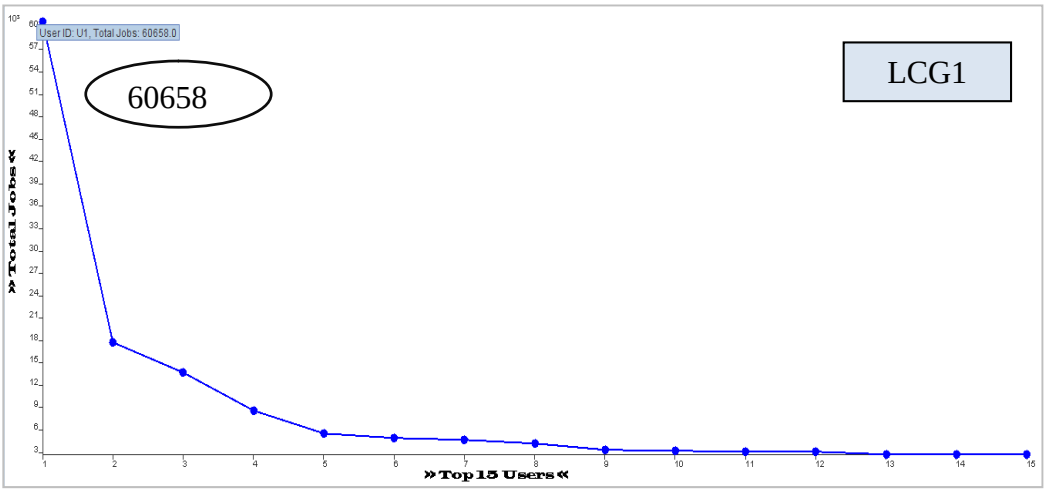

Figure 4.7(a): Top 15 users for LCG1

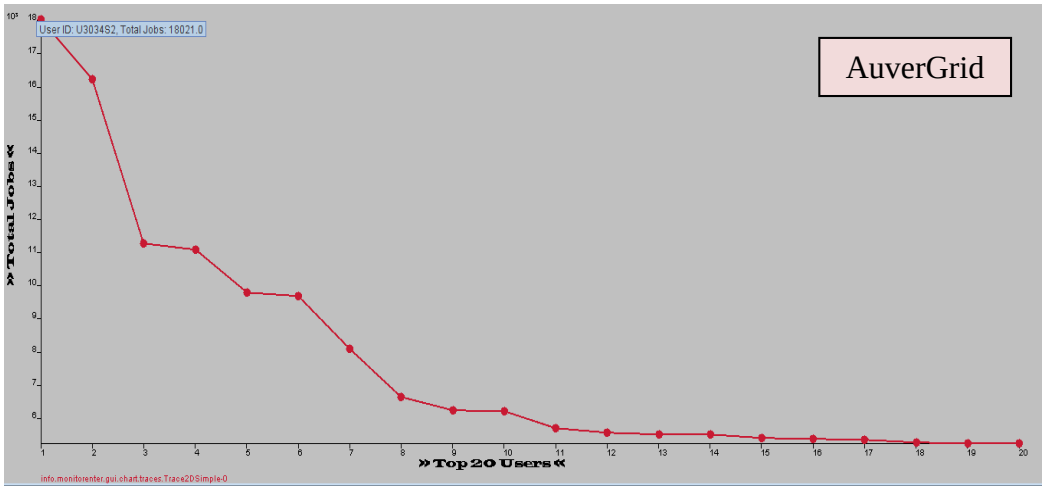

Figure 4.7(b): Top 20 users for AuverGrid

Figure 4.7(a, b) shows the magnitudes of the top 15 and top 20 users for LCG1 and AuverGrid respectively.

User ‘U1’ is the topmost user in LCG1, who submitted ‘60658’ jobs to the system for execution. While ‘U2’, ‘U15’, ‘U15’, ‘U19’ are the next top ranked users with *17697*, *13624*, *8449*, *5485* jobs respectively.

User ‘U3034S2’ is the top most user of AuverGrid, who submitted ‘18021’ jobs to the system for execution. Following next top ranked user are ‘U3034S2’, ‘U247’, ‘U45’, ‘U256’, ‘U257’ with *18021*, *16218*, *11259*, *11083*, *9781* jobs respectively.

4.7.2 Groups Characteristics

Secondly, this research has analyzed the grid workloads from groups (virtual organizations) perspective. SyedWSim has produced the graphs to portray the statistics for the demands of various groups belonging to the LCG1 and AuverGrid grids.

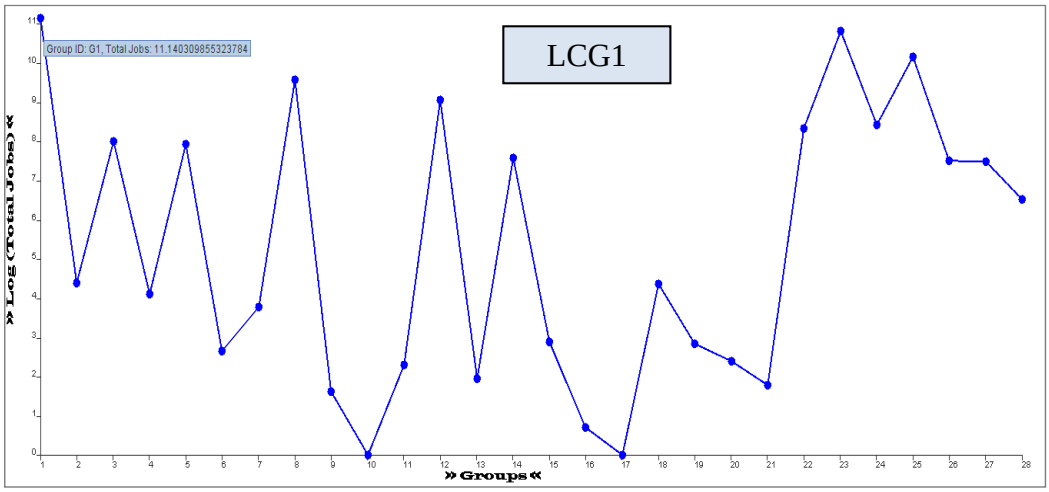

Figure 4.8(a): The Group jobs for LCG1

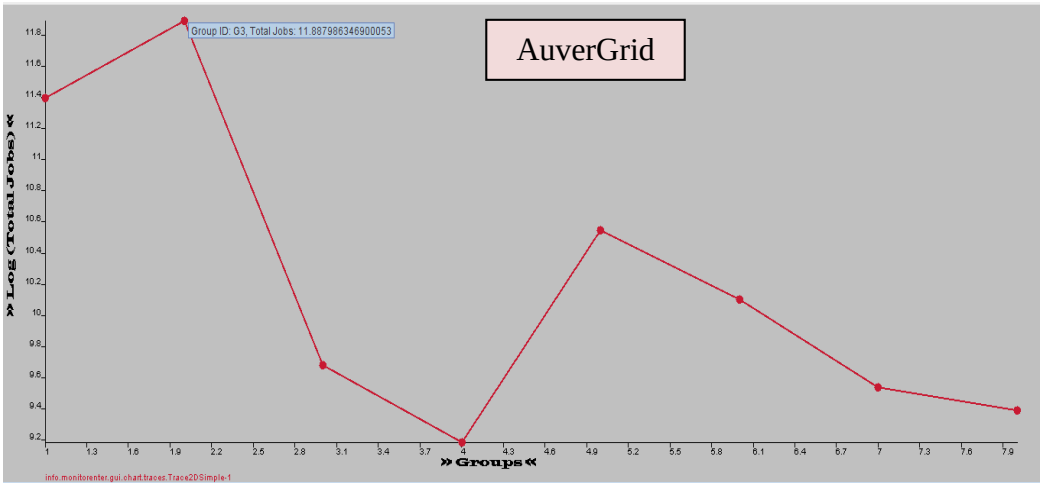

Figure 4.8(b): The Group jobs for AuverGrid

Figure 4.8(a, b) shows the number of jobs per group for LCG1 and AuverGrid respectively. Group ‘G1’ in LCG1 had submitted the maximum number of jobs for execution (i.e., 68893).

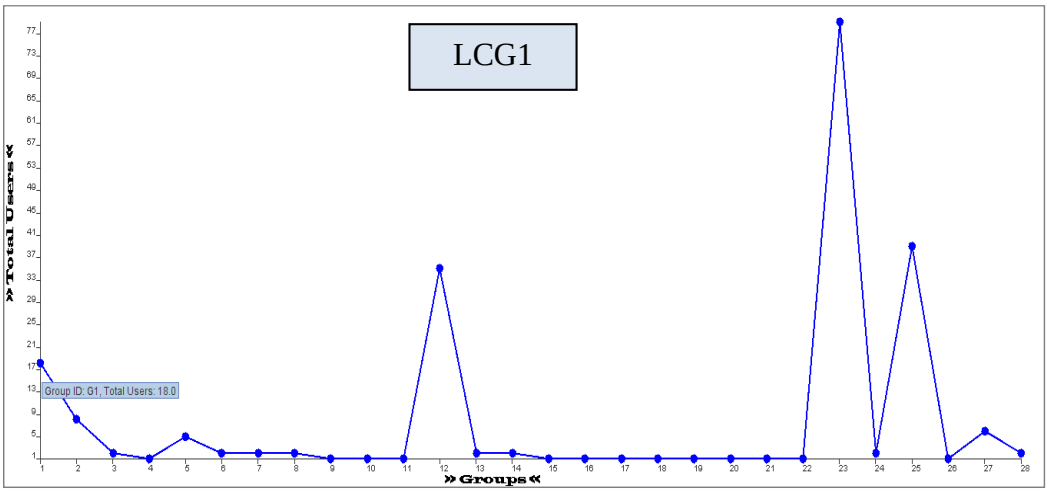

Figure 4.9(a): Groups versus Number of Users for LCG1

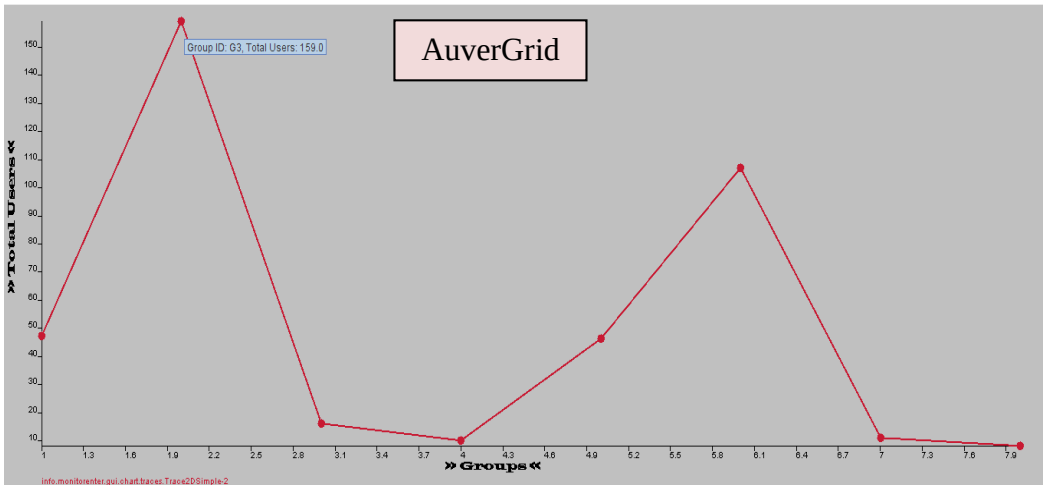

Figure 4.9(b): Groups versus Number of Users for AuverGrid

Figure 4.9(a, b) shows the number of users for each group. Group ‘G1’ in LCG1 had ‘18’ number of users. While Group ‘G3’ in AuverGrid had the ‘159’ number of users.

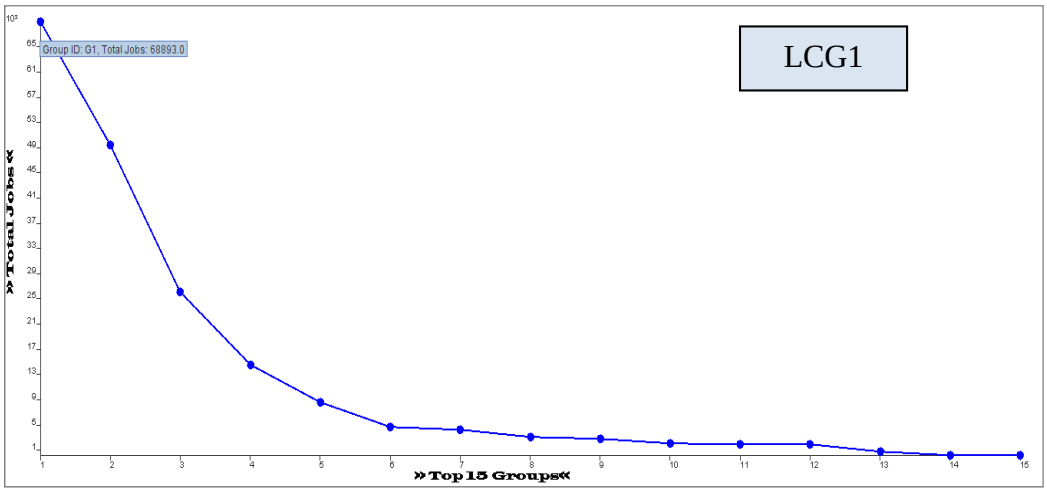

Figure 4.10(a): Top 15 Groups for LCG1

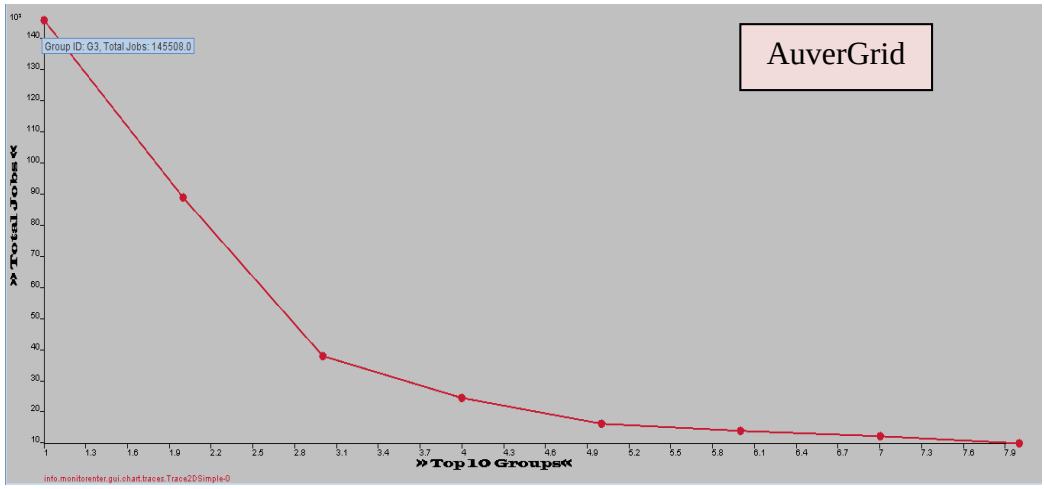

Figure 4.10(b): Top 20 Groups for AuverGrid

Figure 4.10(a, b) shows the magnitudes of the top 15 and top 20 groups for LCG1 and AuverGrid respectively.

Group ‘G1’ in LCG1 had submitted the maximum number of jobs for execution (i.e., 68893). While ‘G4’, ‘G6’, ‘G16’, ‘G2’ are the next top ranked groups with *49292*, *25993*, *14372*, *8477*, *4563* jobs respectively.

While group ‘G3’ is the top most group of AuverGrid, who submitted ‘145508’ jobs to the system for execution. Following next top ranked groups are ‘G4’, ‘G2’, ‘G1’, ‘G7’ and ‘G8’ with *88681*, *37792*, *24311*, *15924*, *13790*, *11903* jobs respectively.

4.7.3 Grid Jobs Characteristics

Finally, SyedWSim has been applied to characterize the dynamic nature of jobs and also make in-depth study of LCG1 and AuverGrid grids.

Count process is introduced to describe job arrival. Count process is formed by dividing the time axis into equally spaced contiguous intervals of time 'T' to produce a sequence of job counts. The values '64' and '128' seconds have been taken as inputs for Time interval (i.e.; 'T') for analysis of LCG1 and AuverGrid traces respectively.

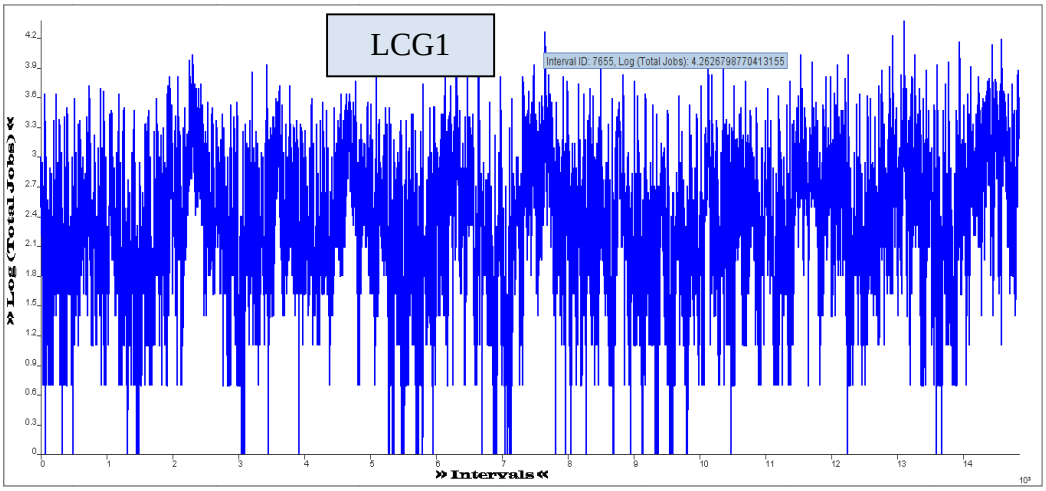

Figure 4.11(a): Job counts for LCG1

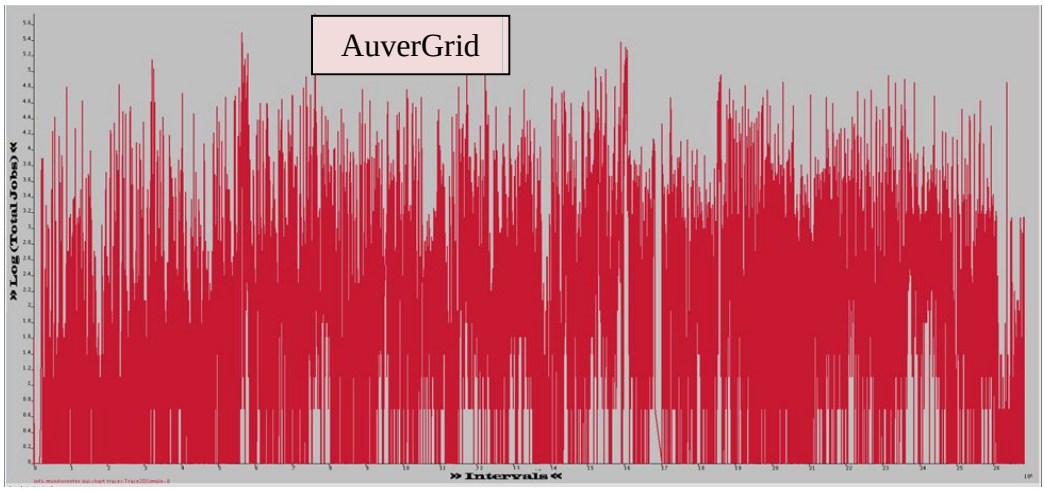

Figure 4.11(b): Job counts for AuverGrid

Figure 4.11(a, b) shows the distribution of job counts for the LCG1 and AuverGrid traces. This Figure shows the continues and random pattern of the job counts.

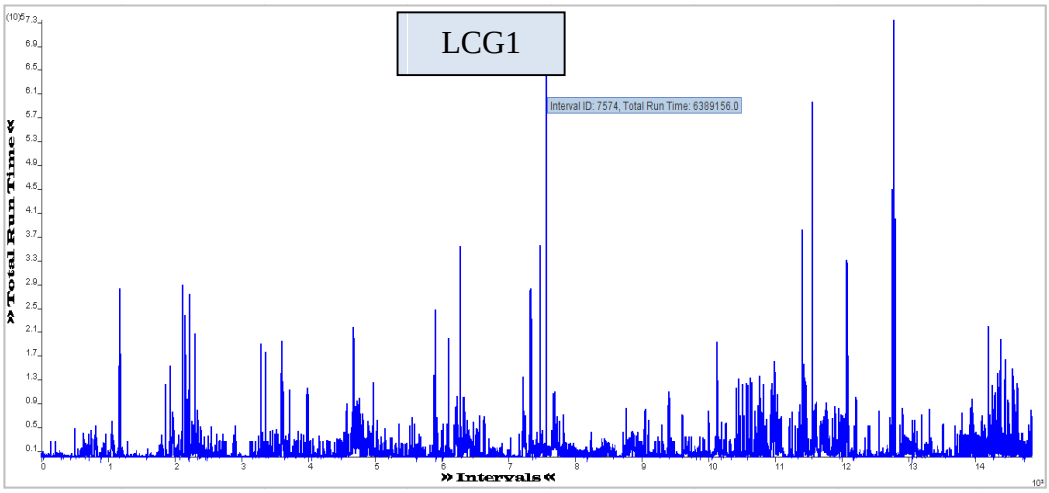

Figure 4.12(a): Total runtime per period for LCG1

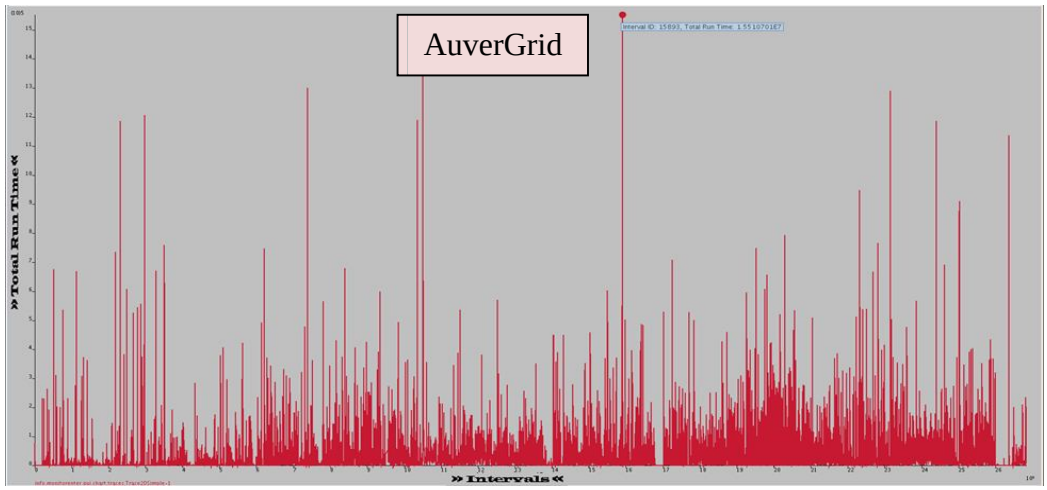

Figure 4.12(b): Total runtime per period AuverGrid

Figure 4.12(a, b) shows the *CPU runtime* demand for the LCG1 and AuverGrid traces. For LCG1, the *CPU runtime* is observed in the range of *'0'* seconds to *'510461'* seconds; with an average value of '*4632.86'* seconds.

While the *CPU runtime* is seen in the in the range of *'0'* seconds to *'1575814'* seconds; with an average value of *'25186.27'* seconds in the AuverGrid.

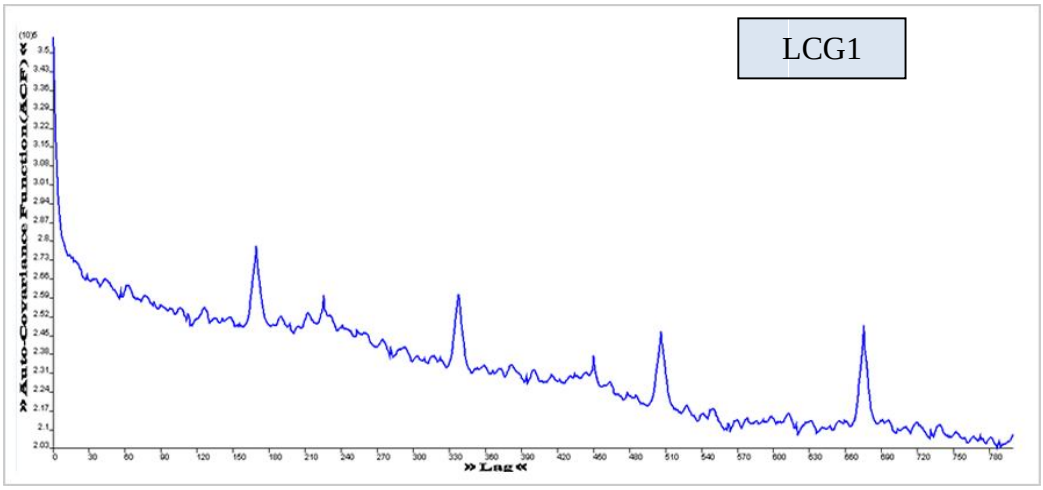

Figure 4.13(a): The autocorrelation function(ACF) of the job counts - LCG1

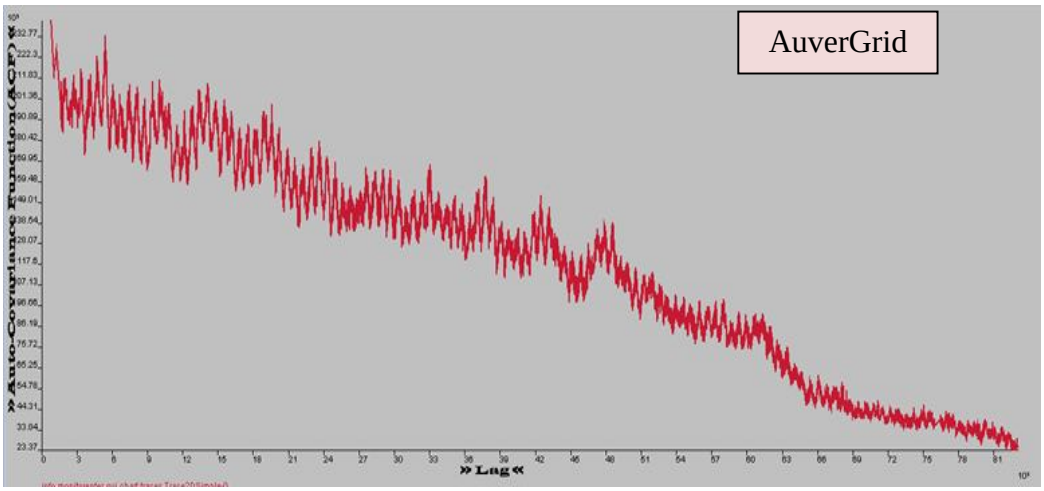

Figure 4.13(b): The autocorrelation function(ACF) of the job counts - AuverGrid

Autocorrelation of a process describes the correlation between different data points in time *'T'*. An autocorrelation of the job counts at different lags has been performed. Figure 4.12(a, b) shows the autocorrelation plots for LCG1 and AuverGrid.

A number of grid job characteristics have been observed in the Figure 4.13 (a, b). Periodicity is clearly detected by the equally spaced peaks in the ACF plot. The decay in ACF with respect to time interval is not exponential and exhibit slow decay, preserving the same pattern in the regular intervals shows the long range dependencies. This behavior can also be analyzed in frequency domain for short range dependency. The strength of regular interval peaks can also be analyzed in the magnitude of Fourier co-efficient. Figure 4.13 (a, b) has shown the correlation structure and periodicity.

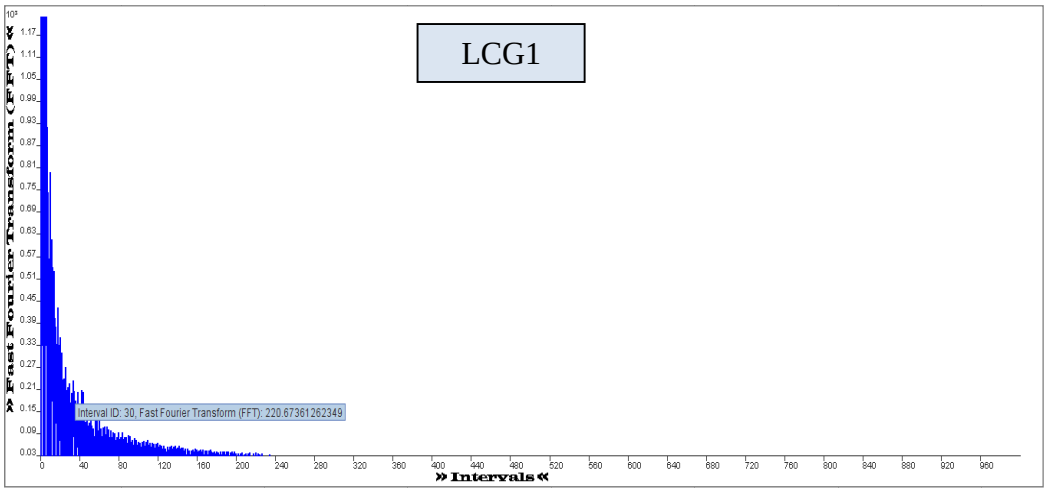

Figure 4.14(a): Fast Fourier transformation(FFT) applied to the autocorrelation of job counts - LCG1

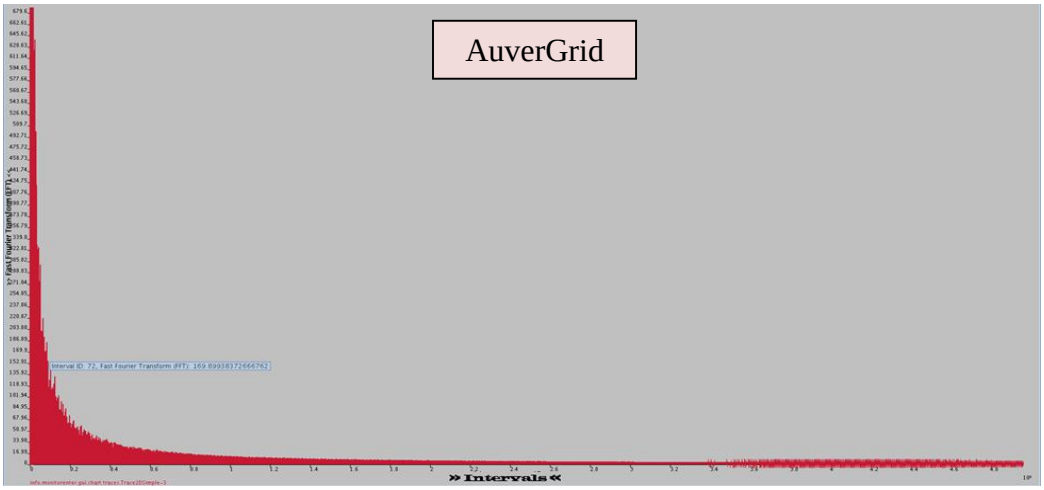

Figure 4.14(b): Fast Fourier transformation(FFT) applied to the autocorrelation of job counts for AuverGrid

Fourier analysis has been applied by applying the FFT on the values of the autocorrelation output. This is shown in Figure 4.14(a, b). This Figure has shown the peaks in the frequency domains. FFT has been applied on the workload traces to identify whether the LCG1 and AuverGrid posses the properties of short range dependencies or not. Multiple harmonics in power spectrum as observed in FFT graph. Figure 4.14(a, b) shows that both workload also posses the properties of short range dependencies.

In summary, Figures 4.11(a, b) to 4.14(a, b) indicate that job arrivals show a diversity of correlation structures, including short range dependency, pseudo periodicity, and long range dependence. Long-range dependence can result in a large

performance degradation, whose effects should be taken into consideration for evaluation of scheduling algorithms. The real grid workloads LCG1 and AuverGrid have shown rich correlation and scaling behavior, which are different from conventional parallel workloads and cannot be captured by simple models such as Poisson or other distribution based methods. LCG1 and AuverGrid will play a key role in the performance evaluation of scheduling algorithms.

4.8 Chapter Summary

This chapter presents a web-based simulator for analysis of grid workload traces. The analysis of real workload traces, from scientific grids, can aid in a wide variety of parallel processing research. For experiments, two multi-clusters grid environments have been analyzed, AuverGrid and the LCG1, using SyedWSim. A thorough analysis has been conducted to study the nature of real workload traces. This simulator allows the user to analyze any real workload trace if it is in the grid Workload Format. Real workload traces play an important role in future grid scheduling studies. Real workload traces have been used as input to the scheduling simulator for performance evaluation of grid resource allocation and job scheduling algorithms, and the details of simulation and experimental results have been thoroughly explained in Chapter 5 and Chapter 6 respectively.

CHAPTER 5

PERFORMANCE ANALYSIS OF GRID RESOURCE ALLOCATION METHODS

5.1 Chapter Overview

Chapter 3 has proposed a new method, the Modified Least Cost Method (MLCM), for efficient and effective utilization of grid resources. This chapter has evaluated the performance of proposed MLCM and other resource allocation methods using simulation on synthetic and real workload traces. To facilitate this research, a simulator has been developed which has produced a comprehensive simulation of a number of grid resource allocation methods.

The structure of the chapter is as follows: Section 5.2 describes the baseline approaches for resource allocation on a grid. Theoretical performance analysis of the grid resource allocation methods is thoroughly explained in section 5.3. Section 5.4 describes the design and development of simulator. Section 5.5 focuses on the experimental results and a discussion and section 5.6 concludes the chapter.

5.2 Baseline Approaches

Following widely used approaches have been considered and simulated for the comparative performance analysis with the proposed MLCM:

- Min-Min algorithm [99], [100], [101] , [102]
- Max-Min algorithm [99], [100], [101] , [102]
- Vogel Approximation method (VAM) [110]

- First Come First Served (FCFS) [110]
- Divisible Load Theory method (DLT) [108]

5.3 Theoretical Performance Analysis of Grid Resource Allocation Methods

This sections describes a theoretical performance analysis of proposed method, MLCM, with respect to other grid resource allocation methods, using resource allocation scenarios *I*.

The resource allocation scenario *I* has been formulated as per Table 3.1. All of the variable values for job workload (A_i), processor availability (B_j) and allocation costs (C_{ij}) are fairly inserted.

5.3.1 Grid Resource Allocation Scenario *I*

In this scenario, the grid consists of six processors (resources) namely P_1, P_2, P_3, P_4, P_5 and P_6 with six jobs (sources) J_1 , J_2 , J_3, J_4, J_5 and J_6 trying to utilize the grid. The workload demands are shown in Table 5.1.

Table 5.1: Workload Demands

Jobs (*J*)	J_1	J_2	J_3	J_4	J_5	J_6
Workload (*A*)	66	55	45	40	50	70

The processor capacities are represented in Table 5.2.

Table 5.2: Processor Capacities

Processors (*P*)	P_1	P_2	P_3	P_4	P_5	P_6
Processor Availability (*B*)	80	70	65	50	80	100

The Resource Allocation Table 5.3 has been formulated from Table 5.1 and 5.2.

Table 5.3: Resource Allocation Scenario I

J\P	P_1	P_2	P_3	P_4	P_5	P_6	***A***
J_1	15	17	19	21	23	14	66
J_2	11	9	13	8	14	16	55
J_3	18	17	22	8	25	10	45
J_4	16	14	12	9	18	13	40
J_5	22	18	13	13	11	15	50
J_6	33	16	17	24	20	10	70
B	80	70	65	50	80	100	

To recap, the allocation cost (C_{ij}) is the cost associated with allocating one unit of processing from job *i* to processor *j*. These are also shown in Table 5.3.

The jobs have workloads of *66*, *55*, *45*, *40*, *50* and *70*, as shown in column '*A*'. The processors have capacities of *80*, *70*, *65*, *50*, *80* and *100*, as shown in row '*B*'. Formally, the grid can be specified by:

m=6
n=6
A [6] = {6 6, 55, 45, 40, 50, 7 0}
B [6] = {8 0, 70, 65, 50, 80,10 0}
C[6][6]={{ 15, 17, 19 , 21, 23, 14 } ,{11 ,9 ,13 ,8 ,14 ,16 }, {18 ,17 , 22, 8, 25 , 10}, {16, 14, 1 2, 9, 18, 13}, {22, 18, 13, 13 , 11, 15}, {33, 16, 17, 24, 20 , 10}}

The problem is to find out which tasks should be allocated to which processors so as to minimize the overall computational cost. There are a number of methods available to find out the computational cost. Firstly, new MLCM will be used to solve the problem. Then, various other resource allocation methods will be applied to solve it.

5.3.1.1 Modified Least Cost Method

Table 5.4 shows the detailed resource allocation using the MLCM for scenario *I* given in Table 5.3.

Table 5.4: Resource Allocation by MLCM

J\P	P_1	P_2	P_3	P_4	P_5	P_6	***A***
J_1	15 **(36)**	17	19	21	23	14 **(30)**	66
J_2	11	9 **(50)**	13	8 **(5)**	14	16	55
J_3	18	17	22	8 **(45)**	25	10	45
J_4	16	14	12 **(40)**	9	18	13	40
J_5	22	18	13	13	11 **(50)**	15	50
J_6	33	16	17	24	20	10 **(70)**	70
B	80	70	65	50	80	100	

MLCM is the iterative procedure. The detailed steps of getting from Table 5.3 to Table 5.4 by applying the MLCM procedure given in appendix *'A'* are as follows:

1. MLCM searches for the least cost value from each row and column of the allocation matrix. In the first iteration, the least cost value is *'8'*.
2. MLCM selects the lowest value from allocation matrix i.e. *'8'*. As *'8'* occurs at two places; MLCM favors selection of the cell which does not include the next least cost value in its corresponding row/column. A tie appears for this case. Now MLCM breaks this tie by selecting the cell *(3, 4)* which can host minimum workload. *'45'* units of J_2 are allocated to P_4.
3. MLCM reduces the processor availability and workload demand by the allocated value i.e. *'45'* units
4. In the second and subsequent iterations, MLCM then searches for the next least allocation cost cell, i.e. cell (2, 4), corresponding to J_2 and P_5. Five units of J_2 are allotted to it. *'50'* units of J_2 are assigned to P_2 and *'70'* units of J_6

are allotted to P_6. Thirty units of J_1 are allocated to P_6 and *'40'* units of J_4 are assigned to P_3. Then, the remaining *'36'* units of J_1 are allotted to P_1.

5. The overall cost is calculated by using *equation (3.1)* as follows:

   15x36 + 14x30 + 9x50 + 8x5 + 8x45 + 12x40 +11x50 + 10x70 =3540

Following are the selected least cost cells (*Job Id, Processor Id*) for the job allocations by applying the MLCM:

{(3,4),(2,4),(2,2),(6,6),(5,5),(4,3),(1,6),(1,10)}

Table 5.5: Processor Allotment by MLCM

Jobs	**Tasks**	**Processor Allotted**
J_1	T_{11}	P_6
	T_{12}	P_1
J_2	T_{21}	P_2
	T_{22}	P_4
J_3	T_{31}	P_4
J_4	T_{41}	P_3
J_5	T_{51}	P_5
J_6	T_{61}	P_6

Table 5.6 shows the mapping of the tasks, of each job, to processors. Job J_1 is divided into two tasks; i.e. T_{11} and T_{12}. The tasks T_{11} and T_{12} are mapped to processors P_6 and P_1 respectively. Job J_2 is also divided into two tasks; i.e. T_{21} and T_{22}. While tasks T_{21} to T_{22} are assigned to processors P_2 and P_4 respectively.

T_{31}, T_{41}, T_{51} and T_{61} are tasks of the Jobs J_3, J_4, J_5 and J_6 respectively. Tasks T_{31} to T_{61} are mapped to P_4, P_3, P_5 and P_6 respectively. For this resource allocation scenario, each job's demand is met using the MLCM. The allocation sequence of each task to a processor by the MLCM is shown below:

$T_{31} \rightarrow P_4$, $T_{21} \rightarrow P_4$, $T_{22} \rightarrow P_2$, $T_{61} \rightarrow P_6$, $T_{61} \rightarrow P_6$, $T_{51} \rightarrow P_5$, $T_{41} \rightarrow P_3$, , $T_{11} \rightarrow P_6$, $T_{12} \rightarrow P_1$

5.3.1.2 Min-Min Method

The Min-Min method begins with the set of all unmapped jobs. Next, row minima (e.g. R_{min}) are computed by taking the least allocation cost value from each row of the resource allocation Table 5.3. Next, the job with the overall minimum allocation cost from R_{min} is selected. Following this, the job workload (i.e. task) is assigned partially or fully to the corresponding available processor depending upon the processor availability at run time. Last, the newly mapped task is removed from the task list, and the procedure repeats until all tasks are mapped [103], [104], [105]. Table 5.6 shows the detailed resource allocation by using the Min-Min method.

Table 5.6: Resource Allocation by Min-Min

J\P	***P*$_1$**	***P*$_2$**	***P*$_3$**	***P*$_4$**	***P*$_5$**	***P*$_6$**	***A***
***J*$_1$**	15 **(36)**	17	19	21	23	14 **(30)**	66
***J*$_2$**	11	9 **(5)**	13	8 **(50)**	14	16	55
***J*$_3$**	18	17 **(15)**	22	8	25	10 **(30)**	45
***J*$_4$**	16	14	12 **(40)**	9	18	13	40
***J*$_5$**	22	18	13	13	11 **(50)**	15	50
***J*$_6$**	33	16	17	24	20	10 **(70)**	70
B	80	70	65	50	80	100	

The detailed procedural steps describing the resource allocation by Min-Min are as follows:

1. Min-Min searches for the minimum value from each row of the allocation matrix. In the first iteration, rows minima are *14, 8, 8, 9, 11* and *10* respectively.
2. Min-Min selects the lowest value from row minima i.e. *'8'*. *'8'* occurs at two places. Min-Min favors selection of the cell which can host maximum workload i.e. *(2, 4)*. *'50'* units of J_2 are allocated to P_5.
3. Min-Min reduces the processor availability and workload demand by the allocated value i.e. *'50'* units
4. In the next and subsequent iterations, Min-Min then searches for the unallocated cells with the least allocation cost from each row of the allocation matrix. Then, Min-Min selects the least value from row minima and allocates the job to the processor while satisfying the scheduling constraints as discussed in chapter 3.

Following are the selected least cost cells (*Job Id, Processor Id*) for job allocation by applying the Min-Min method:

{(2, 4), (2, 2), (3, 6), (6, 6), (5, 5), (4, 3), (1, 1),(6, 2)}

The total computational cost using the Min-Min algorithm for the given resource allocation scenario is *'3705'* units.

5.3.1.3 Max-Min Method

The Max-Min method is very similar to Min-Min. It also begins with the set of all unmapped jobs. Next, the row minima (e.g. R_{min}) are computed by selecting the least allocation cost value from each row of the resource allocation Table 5.3. Next, the job with the overall maximum allocation cost from R_{min} is selected. Following this, the job workload (i.e. task) is allocated to the corresponding processor depending upon the processor availability at run time. Last, the newly mapped task is removed from the task list, and the process repeats until all tasks are mapped [103], [104], [105].

Table 5.7: Resource Allocation by Max-Min

J\P	P_1	P_2	P_3	P_4	P_5	P_6	***A***
J_1	15	17	19	21	23	14 **(66)**	66
J_2	11 **(45)**	9	13	8 **(10)**	14	16	55
J_3	18 **(11)**	17 **(34)**	22	8	25	10	45
J_4	16	14	12	9 **(40)**	18	13	40
J_5	22	18	13	13	11 **(50)**	15	50
J_6	33	16 **(36)**	17	24	20	10 **(34)**	70
B	80	70	65	50	80	100	

Table 5.7 shows the detailed resource allocation using the Max-Min method to solve the resource allocation problem given in Table 5.3.

The detailed procedural steps describing the resource allocation using Max-Min are as follows:

1 Max-Min searches for the minimum value from each row of the allocation matrix. In the first iteration, rows minima are *14, 8, 8, 9, 11* and *10* respectively.

2 Max-Min selects the maximum value from row minima i.e. *'14'*.

3 Max-Min allocates *'66'* units of J_1 to P_6.

4 Max-Min reduces the processor availability and workload demand by allocated value i.e. *'66'* units

5 In the next and subsequent iterations, Max-Min searches for the unallocated cell with the least allocation cost from each row of the allocation matrix. Max-Min then selects the maximum value from row minima and continues allocation, and so on.

Following are the marked cells (*Job Id*, *Processor Id*) for job allocation by applying the Max-Min algorithm:

{(1, 6), (5, 5), (6, 6), (6, 2), (4, 4), (2, 4), (2,2),(2,1), (3,1)}

The total computational cost by the Max-Min algorithm for the given resource allocation scenario is *'4067'* units.

The TORA optimization tool has been used to compute the allocation cost for resource allocation scenario *I*. Procedural steps for the LCM are similar to the Max-Min algorithm. The LCM also yields the same computational cost i.e. *'4067'* units.

5.3.1.4 Vogel's Approximation Method

This section describes the resource allocation by Vogel's Approximation Method (VAM) [109], [110], . Table 5.8 shows the detailed resource allocation using VAM to solve grid resource allocation scenario given in Table 5.3.

Table 5.8: Resource Allocation by VAM

J\P	P_1	P_2	P_3	P_4	P_5	P_6	***A***
J_1	15 **(26)**	17 **(15)**	19	21	23	14 **(25)**	66
J_2	11	9 **(55)**	13	8	14	16	55
J_3	18	17	22	8 **(40)**	25	10 **(5)**	45
J_4	16	14	12	9 **(10)**	18 **(30)**	13	40
J_5	22	18	13	13	11 **(50)**	15	50
J_6	33	16	17	24	20	10 **(70)**	70
B	80	70	65	50	80	100	

The procedural steps to solve the resource allocation scenario *I* are as stated below:

1. VAM computes row penalties and column penalties:
    a. Row penalties are computed by subtracting the least cost value from the next least cost value in the same row. In the first iteration, row penalties are as follows: *{1, 1, 2, 3, 2, 6}*
    b. Column penalties are computed by subtracting the least cost value from the next least cost value in the same column. In the first iteration, column penalties are as follows:*{4, 5, 1, 0, 3, 0}*
2. VAM selects the maximum value from row penalties and column penalties i.e. *'6'* corresponding to J_6.
3. VAM allocates the *'70'* units of J_6 to P_6 and reduces the processor availability (*B*) and workload demand (*A*) by *'70'* units.
4. In the second and subsequent steps, VAM computes the row penalties and column penalties for resource allocation matrix and continues allocation.

The total computational cost by VAM for the given resource allocation scenario *I* is *'3740'* units.

5.3.1.5 Divisible Load Theory

This section presents the computational results using the Divisible Load Theory (DLT) method as described in [108].

DLT method divides the load equally into portions, each of which can be allocated to a separate processor. DLT method is used for fair and equal distribution of load among grid nodes. A number of of methods have been introduced for the division of a load, from multiple sources into equal amounts. Random number method has been used in [108] for the distribution of load.

For DLT method another notation (i.e. $S_{i)}$ is introduced which denotes the number of partitions of Job *i*. The detailed resource allocation by the DLT method is shown in Table 5.9.

Table 5.9: Resource Allocation by DLT

J\P	P_1	P_2	P_3	P_4	P_5	P_6	***A***	***S***
J_1	15 **(22)**	17 **(22)**	19	21	23	14 **(22)**	66	3
J_2	11 **(11)**	9 **(11)**	13 **(11)**	8 **(11)**	14 **(11)**	16	55	5
J_3	18	17 **(15)**	22	8 **(15)**	25	10 **(15)**	45	3
J_4	16	14 **(10)**	12 **(10)**	9 **(10)**	18	13 **(10)**	40	4
J_5	22	18	13	13 **(25)**	11 **(25)**	15	50	2
J_6	33	16	17 **(35)**	24	20	10 **(35)**	70	2
B	80	70	65	50	80	100		

The procedural steps of the DLT method to solve the grid resource allocation scenario *I* are as detailed below:

1. The DLT method divides the load into portions of equal sizes. J_1 is divided into *'3'* portions; each of the size *'22'* units, J_2 is divided into *'5'* portions and so on.
2. The DLT method selects the first job from the Job list and finds the least unallocated cells from the first row. The DLT allocates *'3'* portions of J_1 to the three available least cost cells i.e. *(1, 6), (1, 1)* and *(1, 2)* respectively.
3. The DLT method reduces the processor availability and workload demand by the allocated values.
4. In next and subsequent iterations, the DLT method selects the next job and finds the available unallocated least cost cells in the job's corresponding row and continues with the allocation.

The total computational cost by the DLT method for the given resource allocation scenario is *'4167'* units.

5.3.1.6 First Come First Served

Table 5.10 shows the detailed resource allocation using FCFS to solve the grid resource allocation scenario given in Table 5.3.

Table 5.10: Resource Allocation by FCFS

J\P	P_1	P_2	P_3	P_4	P_5	P_6	**A**
J_1	15 **(66)**	17	19	21	23	14	66
J_2	11 **(14)**	9 **(41)**	13	8	14	16	55
J_3	18	17 **(29)**	22 **(16)**	8	25	10	45
J_4	16	14	12 **(40)**	9	18	13	40
J_5	22	18	13 **(9)**	13 **(41)**	11	15	50
J_6	33	16	17	24 **(9)**	20 **(61)**	10	70
B	80	70	65	50	80	100	

Detailed steps of FCFS for resource allocations are as follows:

1 FCFS selects the first job from the Job list. It matches the job demand with the first available processor from the available processor list. For example, the demand of J_1 is *'66'* units which can be satisfied by the P_1.

2 FCFS allocates *'66'* units of J_1 to P_1.

3 FCFS reduces the processor availability and workload demand by the allocated value i.e. *'66'* units. Then, FCFS selects the Job J_2. *'14'* units of J_2 are allocated to P_1 while the remaining *'41'* units of J_2 are assigned to P_2. Then, FCFS selects the next job from the Job list and finds the available processor for allocation and so on.

The total computational cost using the FCFS algorithm for the given resource allocation scenario is *'4924'* units.

The allocation cost to solve the resource allocation scenario I is also computed by the North West Corner method (NWCM) (Taha, 2002) using the TORA optimization tool. Procedural steps for NWCM are similar to the FCFS resource allocation method. NWCM also yields the same computational cost i.e. *'4924'* units.

The overall computational cost of using each method to solve resource allocation scenario I is shown in Table 5.11.

Table 5.11: Performance Results of Resource Allocation Methods for Scenario I

Resource Allocation Method	**Computational Cost**
MLCM	3540
Min-Min	3705
Max-Min	4067
VAM	3740
DLT	4167
FCFS	4924

5.4 Resource Allocation Simulator Design and Development

In order to evaluate the effectiveness of our approach, a resource allocation simulator is developed. The simulation software comprises two parts. One part takes grid resource allocation scenario as input. Each scenario is described by values of m, n, A, B and $C_{ij.}$

The second part of the simulation solves the resource allocation problem using different grid resource allocation methods. In this software; MLCM, Min-Min, Max-Min, Vogel's Approximation Method (VAM), DLT and FCFS methods have been programmed for simulation and evaluation purposes.

Developed simulator produces the comprehensive simulation for each resource allocation method. The developed simulator displays the mapping of job(s) to processor(s) at each step during resource allocation. The stepwise execution of the MLCM to solve grid resource allocation scenario *I* is shown in Table 5.12.

Table 5.12: Stepwise execution of MLCM for Scenario I

Step No.	**Least Cost Cell**	**Least Cost Value**	**Allocated Job Workload**	**Remaining Job Demand**	**Processor Availability**
1	(2, 3)	8	45	0	5
2	(1, 3)	8	5	50	0
3	(1, 1)	9	50	0	20
4	(5, 5)	10	70	0	30
5	(4, 4)	11	50	0	30
6	(3, 2)	12	40	0	25
7	(0, 5)	14	30	36	0
8	(0, 0)	15	36	0	44

The simulator shows the least cost cell for each iteration. It also shows how much workload has been allocated to a processor in every step during job execution. It also shows remaining job demand and processor availability at each iteration. Total computational cost computed by the simulator for Grid resource allocation scenario *I* is *'3540'*.

It should be noted that only job metadata is passed to the simulator. The simulator processes the metadata for the list of jobs that have been assigned to it. The simulator is written in Java and makes use of the 'MPJ express' API.

5.5 Performance analysis and evaluation

5.5.1 Simulation Data

All grid resource allocation methods have been tried with several grid resource allocation scenarios. A number of resource allocation scenarios have been produced using the Monte Carlo method except scenario *II*. The resource allocation scenario *II* is taken from [108]. In scenarios *I* to *VIII*, the Monte Carlo method has been applied to fairly distribute the numbers to workload demands (i.e. A_i), processor availabilities (i.e. B_j) and the costs of allocation (i.e. C_{ij}). Moreover, scenarios *IX* to *XII* have used real workload traces of LCG1. All of these scenarios have been used in simulation for performance evaluation of different grid resource allocation methods.

5.5.2 Grid Resource Allocation Scenario I

In this resource allocation scenario, the grid is defined by *'6'* jobs and *'6'* processors. Input specifications for the simulator are as follows:

m=6

n=6

A [6] = {66, 55, 45, 40, 50, 70}

B [6] = {80, 70, 65, 50, 80, 100}

C[6][6]={{15, 17, 19, 21, 23, 14}, {11, 9, 13, 8, 14, 16},
{18, 17, 22, 8, 25, 10}, {16, 14, 12, 9, 18 ,13},
{22, 18, 13, 13, 11, 15}, {33, 16, 17, 24, 20, 10}}

5.5.3 Grid Resource Allocation Scenario II

In this scenario, the grid consists of five processors with four jobs. Input specifications for the simulator are shown below:

m=4

n=5

A[4] = {6, 9, 6, 8}

B[5] = {12, 12, 12, 12, 12}

C[4][5]={{8, 9, 2, 8}, { 8, 8, 9, 2} , {8, 8, 8, 9},{2, 8, 8, 8}, {9, 2, 8, 8}}

5.5.4 Grid Resource Allocation Scenario III

In this resource allocation scenario, the grid is defined by *'8'* jobs and *'8'* processors. Input specifications for the simulator are as follows:

m=8

n=8

A [8] = {70, 80, 55, 60, 60, 75, 45, 80}

B [8] = {90, 95, 65, 60, 75, 90, 55, 25}

C[8][8]={{15,16,19,21,23,14,11,9},{13,8,14,16,18,17,22,13},
{25,10,16,14,12,9,8,13},{22,9,13,13,11,15,33,10},
{17,24,20,10,15,10,19,21},{9,14,11,9,13,8,12,16},
{18,16,22,8,22,10,16,9}, {12,9,18,13,24,7,13,13}}

5.5.5 Grid Resource Allocation Scenario IV

In this scenario, the grid consists of *'16'* jobs and *'8'* processors. The costs of allocations are taken in the range of *'10'* to *'100'*. Formally, the input specification can be written as:

m=16

n=8

A [16] = {70,80,55,50,60,65,25,90,70,80,55,60,40,45,35,50}

B [8] = {100, 150, 90, 100, 100, 90,150,250}

C[16][8]={{18,90,63,54,36,25,54,57},{74,48,86,72,18,20,54,98},
{11,33,23,82,26,10,37,67},{12,85,56,35,75,37,37,14},
{62,43,79,16,60,60,18,77},{17,74,62,25,19,52,11,96},
{63,49,66,56,77,18,35,23}, {96,54,25,10,18,38,13,47},
{53,46,74,17,72,35,68,41}, {42,80,18,79,79,40,26,40},
{92,62,86,84,16,67,15,93}, {32,74,88,10,75,14,49,41},

{85,61,60,73,43,68,17,12}, {44,34,70,26,75,64,53,74},
{56,47,28,12,64,38,79,78}, {76,83,74,47,65,63,79,67}}

5.5.6 Grid Resource Allocation Scenario V

This resource allocation scenario is a variant of resource allocation scenario *IV*. Workload demands and processor capacities are identical to the values given in resource allocation scenario *IV*. But the costs of allocations are taken in the range of *'0'* to *'25'* units instead of *'10'* to *'100'*. Formally, the input specification can be written as:

m=16
n=8
A[16] = {70,80,55,50,60,65,25,90,70,80,55,60,40,45,35,50}
B[8] = {100, 150, 90, 100, 100, 90,150,250}
C[16][8]={{9,4,17,15,24,3,10,0},{16,8,5,13,18,24,0,20},{9,8,16,20,23,6,1,0}
{19,17,13,10,2,16,14,9},{3,7,2,17,11,23,12,21},
{20,23,21,4,6,18,12,17},{5,18,6,24,19,6,22,11},
{22,7,4,6,13,18,5,12},{18,9,13,15,24,10,5,10},
{12,18,7,20,20,17,0,6}, {13,7,15,20,24,19,18,16},
{13,18,5,17,8,16,6,11,22},{5,12,11,13,18,14,22,24},
{17,1,8,1,2,17,7,2},[160,3,15,22,11,23],{19,24,6,3,18,21}}

5.5.7 Grid Resource Allocation Scenario VI

In this scenario, the grid consists of *'16'* processors (i.e. *n=16*) and *'50'* jobs (i.e. *m=50*). Processors have capacities in the range of *'50'* to *'200'* units. Workload demand for each job varies from *'10'* to *'40'* units. The costs of allocations (C_{ij}) vary from *'10'* to *'50'* units.

5.5.8 Grid Resource Allocation Scenario VII

In this scenario, the grid consists of *'32'* processors (i.e. *n=32*) and *'100'* jobs (i.e. *m=100*). Processors have capacities in the range of *'150'* to *'500'* units. Workload demand for each job varies from *'20'* to *'100'* units. The costs of allocations are taken in the range of *'10'* to *'50'* units.

5.5.9 Grid Resource Allocation Scenario VIII

This scenario is a variant of grid resource allocation scenario *VII*. In this scenario, the same workload and the cost of allocation have taken as used before in resource allocation scenario *VII*. However, processors have processing capabilities in the range of *'150'* to *'400'* units.

5.5.10 Grid Resource Allocation Scenarios IX-XI

Grid resource allocation scenarios *IX* to *XII* are composed of real workload traces of LCG1. First *'500'*, *'1000'* and *'2000'* jobs of LCG1 have been used as workload in the composition of resource allocation scenarios *IX*, *X* and *XI* respectively. Resource allocation methods have been evaluated using *'32'*, *'64'* and *'128'* number of CPUs. In addition, Monte Carlo method has been applied to fairly distribute the processing capabilities among the processors in the range of *'200'* to *'6900'* units. The costs of allocations (C_{ij}) have been varied from *'10'* to *'59'* units.

5.6 Results and Discussions

Firstly, theoretical analysis of resource allocation methods is performed by taking resource allocation scenario *I*. Then, experimental performance analysis of resource allocation methods is conducted using developed simulator for the same resource allocation scenario *I*. The same computational costs are obtained for each method using theoretical analysis as well as by simulation.

Details experimentation has been performed using developed simulator to solve resource allocation scenarios numbered *II* to *XII* to evaluate the efficiency of proposed MLCM in comparison to other well known grid resource allocation methods. The computed computational costs for each method and for first eight scenarios based on synthetic workload traces are shown in Table 5.12.

Table 5.12: Computational costs using synthetic workload traces

	Resource Allocation Method					
	MLCM	Min-Min	Max-Min	VAM	DLT	FCFS
Scenario I	**3540**	3705	4067	3740	4167	4924
Scenario II	**160**	**160**	202	**160**	162	214
Scenario III	**4735**	**4735**	5880	5245	5754	6125
Scenario IV	**2500**	2915	3590	5260	9875	11750
Scenario V	**8139**	8219	8347	11940	15634	23332
Scenario VI	**20235**	20665	22685	33390	38722	42690
Scenario VII	52824	**52179**	52273	76328	110223	125400
Scenario VIII	**52685**	52914	52824	76676	112154	135958

The MLCM produces best results as compared to all other resource allocation methods for scenario *I*.

The MLCM, Min-Min and VAM produce the same and least computational costs for the resource allocation scenario *II*. MLCM and Min-Min also produce the same and least computational cost for the scenario *III*.

Resource allocation scenario *V* is a variant of resource allocation scenario *IV*. In scenario *IV*; the costs of allocations are in the range of *‘10’* to *‘100’*. While in the resource allocation V, the costs of allocations vary from *‘0’* to *‘25’*. MLCM produces

the best results for both scenarios *IV* and *V* for a different range of costs of allocations.

MLCM produces the best results as compared to other resource allocation methods for resource allocation scenario *VI* as well. The computational costs computed by Min-Min and Max-Min are slightly higher than the values computed by MLCM.

Resource allocation scenario *VIII* is a variant of resource allocation scenario *VII*. In both scenarios, the grid consists of *'32'* processors and *'100'* jobs. In scenario *VII*, processors have more processing capacities i.e. in the range of *'150'* to *'500'* units. While in scenario *VIII*, processing capabilities of processors are in the range of *'150'* to '400' units. Min-Min and Max-Min produce slightly less computational cost as compared to MLCM for scenario *VII*. But, MLCM produces the best computational results as compared to all other resource allocation methods for resource allocation scenario *VIII*. It is evident that MLCM performs well and produces the lower computational cost in comparison to other resource allocation methods even if the grid consists of processors with less processing capability.

First top three base line approaches, i.e., MinMin, MaxMin and VAM have been selected from Table 5.12, and then compared with proposed MLCM using real workload traces of LCG1 under increasing workload and varying the number of CPUs. The computed performance results, for each resource allocation method, for scenarios *IX* to *XI* are shown in Table 5.13.

Table 5.13: Computational costs using real workload traces of LCG1

Scenario	**Workload Size**	**Number of CPUs**	**MLCM**	**MinMin**	**MaxMin**	**VAM**
IX	500	32	**13120101**	13131779	13204195	13123186
X	1000	64	**15088100**	15090821	15094020	15073094
XI	2000	128	**30802089**	30802933	30807914	30801173

Table 5.13 shows that MLCM has shown the best computational cost compared to other resource allocation methods. Min-Min has shown the higher computational cost than MLCM for real workload traces of LCG1. Moreover, MLCM has shown the optimal performance when the number of jobs increased from *'500'* to *'1000'* and then *'2000'* using *'32'*, *'64'* and *'128'* CPUs respectively.

In summary, in eight out of twelve scenarios, MLCM is superior to Min-Min, in two scenarios they are equivalent, and in one scenario, Min-Min is superior. Max-Min also results in less computational cost but shows poor performance as compared to MLCM and Min-Min. The DLT method and FCFS produce the highest computational cost for all resource allocation scenarios. VAM results in less cost for simple resource allocation scenarios but produces high computational cost for complex scenarios. Figure 5.1 shows a graph derived from Table 5.12.

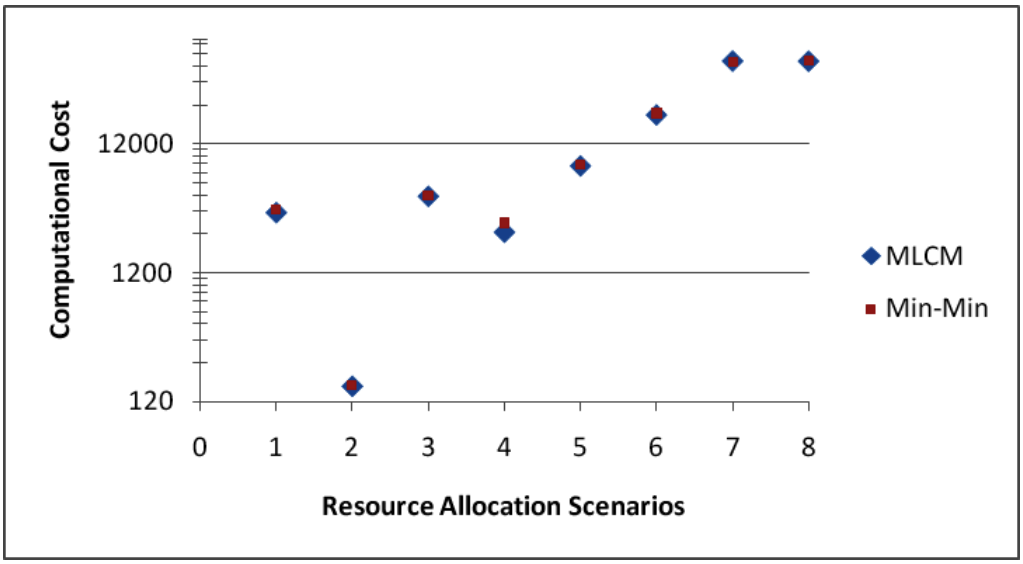

Figure 5.1: Comparison of MLCM and Min-Min

Figure 5.1 shows the comparative performance analysis of MLCM and Min-Min methods. From experimental results, it is evident that MLCM yields the small improvement in computational cost over Min-Min method for a variety of resource allocation scenarios.

5.7 Chapter Summary

This chapter has evaluated the performance of MLCM for a variety of grid workload scenarios. The performance of MLCM was compared with other known ones using simulation for different grid resource allocation scenarios of diverse nature. Experimental results have shown that MLCM yields improvements in terms of performance and results in lower computational cost in terms of time as compared to other resource allocation strategies. It has experimentally proven that MLCM is a promising technique to use in grid environment, when dealing with tasks allocation.

CHAPTER 6

PERFORMANCE ANALYSIS OF JOB SCHEDULING ALGORITHMS

6.1 Chapter Overview

This chapter presents a comparative performance analysis of proposed scheduling algorithms with other widely used job scheduling algorithms. An extensive experimentation have been carried out for evaluation of scheduling algorithms on an experimental grid using synthetic and real grid workload traces, taken from leading computational centers.

This chapter is organized as follows: Section 6.2 presents the baseline approaches considered for the experiment. Section 6.3 highlights the list of proposed scheduling algorithm in this work. The homogenous implementation of new scheduling algorithms is detailed in section 6.4. In section 6.5, the scheduling simulator's design and development are discussed. Section 6.4 describes the detailed performance analysis of the grid scheduling algorithms. Section 6.5 concludes the chapter.

6.2 Base line approaches

Baseline approaches considered for the experiment are as follows:

- First Come First Served (FCFS) [117], [119], [120]
- Shorted Process Next (SPN) [117], [119], [120]
- Longest Job First (LJF) [117], [119], [120]
- Priority(P) [117], [119], [120]
- Round Robin (RR) [50] ,[117], [119], [120]
- Proportional Local Round Robin (PLRR)[82]

- Self Adjustment Round Robin (SARR) [121]
- Intelligent Time slice for Round Robin (NIR) [123]
- Round Robin Priority (NRR) [122]
- A New Multilevel CPU Scheduling algorithm (MR) [124]
- Shortest Remaining Burst Round Robin (SRBRR) [125]

6.3 Proposed Scheduling Algorithms

Proposed scheduling algorithms considered for the experiment are as under:

1. Multilevel Hybrid scheduling algorithms (MH)
2. Multilevel Dual Queue Scheduling algorithms (MDQ)
3. Dynamic Multilevel Hybrid Scheduling Algorithm using Median(MHM)
4. Dynamic Multilevel Hybrid Scheduling Algorithm using square root(MHR)
5. Dynamic Multilevel Dual Queue Scheduling Algorithm using Median (MDQM)
6. Dynamic Multilevel Dual Queue Scheduling Algorithm using Square root (MDQR)

6.4 Homogeneous Implementation of Job Scheduling Algorithms

Master-slave architecture has been employed for implementation of proposed and other job scheduling algorithms, shown in Figure 6.1.

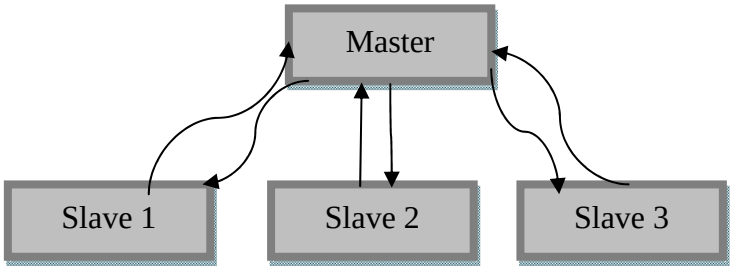

Figure 6.1: Block diagram of master-slave architecture

In this architecture, one processor is dedicated as the master processor among the cluster nodes. The master processor is responsible for distribution of the workload among the slave processors using round robin allocation strategy (i.e. 1, 2, 3…. n, 1) for parallel computation.

The same algorithm, either MHM or MHR, is used on each slave processor. Once the computation is complete, the results are sent to the master processor.

6.5 Scheduling Simulator Design and Development

The MPJ-express is widely used Java based message passing library that allows writing and executing parallel applications for distributed and multicore systems[151].

A Java based simulator has been developed using MPJ-express API to evaluate the efficiency of proposed and other scheduling algorithms. In this approach, the scheduling simulator communicates with grid nodes by message passing. The metadata for each process includes its ID, its arrival time, its CPU time and the number of slaves to choose (jobs have to be divided between them). The simulation software encounters the arrival time for each process and then sends processes to the system. The software has two main programs; first program runs on the master node (SimM) and another program runs on each slave processor (SimS). SimM accepts a workload and distributes among slave processors using RR. SimM receives notification from each slave processor for each job (or part of a job) that has completed. Each slave runs SimS and computes the average waiting time, the average turnaround time, the average response times and other performance parameters. SimS processes the metadata for the list of processes that have been assigned to it. Upon completion of a process, SimM is informed. SimS keeps a detailed record of the processes being executed on each slave (process ID, submit time, CPU time, time quantum, priority, start time and completion time).

All slaves use the same scheduling algorithm that is input by the user of SimM. The user can select one of a range of algorithms including the newly developed ones such as MHM, MHR, MH, MDQ, MDQM and MDQR, and existing ones such as

FCFS, SJF, RR, P etc. The purpose of the simulator is to produce a comparative performance analysis of scheduling algorithms.

6.6 Performance Analysis

This section presents a detailed comparison of proposed scheduling algorithms with the well-known scheduling approaches.

6.6.1 Experimental Setup

The experiments have been performed using High Performance Computing facility at High Performance Computing Center of Universiti Teknologi PETRONAS. The SGI ALTIX 4700 machine has been used as experimental grid. It is a single image system, which is achieved via hardware architecture. The computing blades are inter-connected via NUMA-Link. It consists of 128 cores (64 Dual Core processors) and each blade has 32GB of local memory. The 'hpc.local' was used as the default execution site for job submission. A detailed experimental setup is shown in Table 6.1.

Table 6.1: Experimental Setup

Name	**Type**	**Location**	**Configuration**
gillani	Shell terminal	Lab Workstation	Intel Core 2 Duo CPU 2.0GHZ 2 GB Memory
hpc.local	Execution site	HPC facility	CPUs: 128 cores (64 Dual Core processors) CPU MHz : 1.6 GHz arch : IA-64, EPIC based Model: Itanium2 9030 series Operating System: Suse Linux Enterprise Server 11(SLES 11) Memory: 1Tera Byte Distributed Shared Network connectivity: 100Mbps (local connectivity)

6.6.2 Simulation Data

In the scheduling experiment, a number of complex synthetic and real workload traces have been used to demonstrate the scheduling capability of each algorithm from different perspective. Real workload traces have been taken from the leading computational centers [161], [162] which are publicly available on grid workload archive [34] for the experimental purposes to evaluate the performance of scheduling algorithms. These workload traces facilitates the experimental simulation for this research. The characteristics of the workload traces have been thoroughly discussed in Chapter 4. These real workload traces possess the properties of self-similarity, pseudo periodicity and long-range dependencies. Table 6.2 shows that simulation data.

Table 6.2: Simulation Data

Sl. No	Data Type	Total number of Jobs	Number of CPUs
1	Synthetic workload	1000 jobs	{16, 32, 64}
2	Synthetic workload	2000 jobs	{16, 32, 64}
3	Real workload trace	18804 (10% of LCG1)	{16, 32, 64, 128}
4	Real workload trace	37608(20% of LCG1)	{16, 32, 64, 128}
5	Real workload trace	12125(3% of AuverGrid)	{16, 32, 64, 128}
6	Real workload trace	20208(5% of AuverGrid)	{16, 32, 64, 128}

Two synthetic workload traces (i.e., Sl. No.: 1-2) have been produced using the Monte Carlo method. The CPU burst times have been distributed among workload jobs in the range of '15' to '16787' units. Priorities have been distributed among jobs in the range of '0' to '9'. Moreover, the traces (i.e., Sl. No.: 3-4) are taken from a real Grid workload i.e., 'LCG1' whilst two traces (i.e., Sl. No.: 5-6) have been taken from 'AuverGrid' for the experiment. Priority attribute is missing in the real grid workload traces of LCG1 and AuverGrid[34]. The Monte Carlo method has been applied to inject priorities in workload traces for each job in the range of '0' to '15'. In scheduling experimentation, lowest number indicates the highest priority and vice versa.

6.6.3 Performance Metrics

Performance metrics for evaluation of the grid scheduling algorithms include the average waiting time, average turnaround time, average response time, average slowdown, total completion time and maximum stretch time of the job. The core objective of scheduling algorithm is to minimize the values of each performance parameter to attain the optimal solution. Average response time is the most demanding parameter from the user's perspective whilst other five performance parameters are needed to be minimized from the system's perspective to achieve high computing power for job or application processing.

6.6.4 Results and Discussion

According to proposed performance evaluation strategy, experiments have been performed in three phases on an experimental grid for comparative performance analysis of grid scheduling algorithms.

In the first phase, a series of experiments have been performed for synthetic workload traces of small and medium sized workload by varying the number of CPUs successively from '16' to '64'. The efficiency, scalability and performance have been evaluated in dynamic grid environment. Experimentation used '50' units as a fixed time quantum for this experimental phase.

In the second phase, experiments have been performed using ‘LCG1’. Experimentation includes the efficiency, performance and scalability test of scheduling algorithms under an increased real workload and increased processors availability. Two data sets have been formed, first by using ‘10%’, and second by using ‘20%’ of the LCG1 workload (i.e. 18804, and 37608 processes), respectively. The ‘runtime’ attribute is given for each process in ‘LCG1’. The ‘runtime’ is taken as CPU time in this experiment. A series of experiments have carried on experimental grid by varying the number of CPUs successively from ‘16’ to ‘64’. This experimental phase also used ‘50’ units as the fixed time quantum.

Similarly, in the third phase, a number of experiments have been carried out for various scheduling algorithms using ‘AuverGrid’, a real workload trace. These experiments also include the efficiency, performance and scalability test of scheduling algorithms under an increasing real workload. Two data sets have been formulated, first by using 3%, and second by using ‘5%’ of the AuverGrid workload, i.e., ‘12125’, and ‘20208’ processes, respectively. In this phase, experimentation has also been performed by varying the number of CPUs from ‘16’ to ‘128’. Experimentation has used ‘50’ units as the fixed time quantum.

Furthermore, all job scheduling algorithms have been evaluated in the dynamic grid environment. Dynamic grid environment means all jobs are appearing in the system during simulation will not remain fixed over time as some new jobs are arriving. In the following sections, experimental results from the scheduling experiment on a grid are discussed in details. Details experimental results have also been shown in appendix *‘C’*.

6.6.4.1 Average Waiting Times Analysis

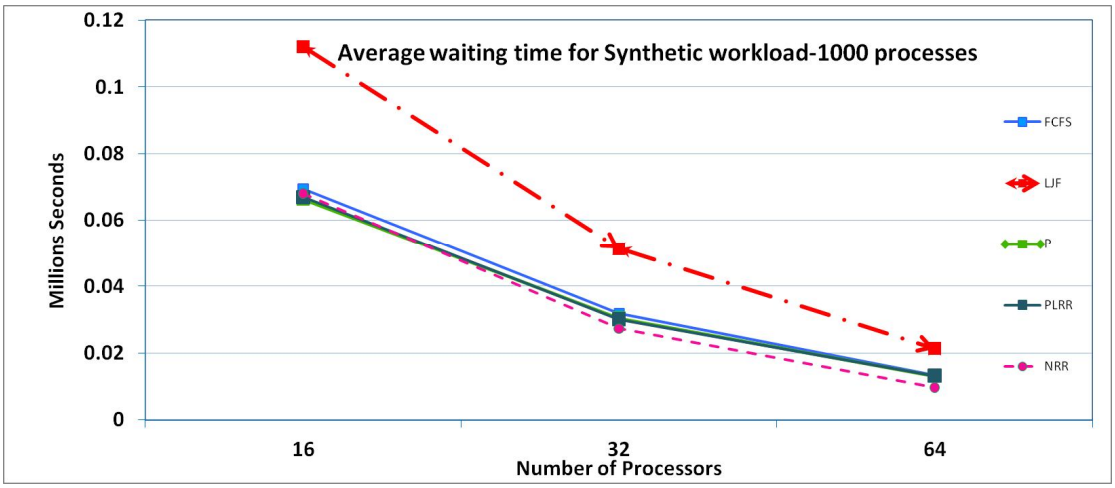

Figure 6.2(a): Average waiting times of five algorithms for synthetic workload of 1000 processes

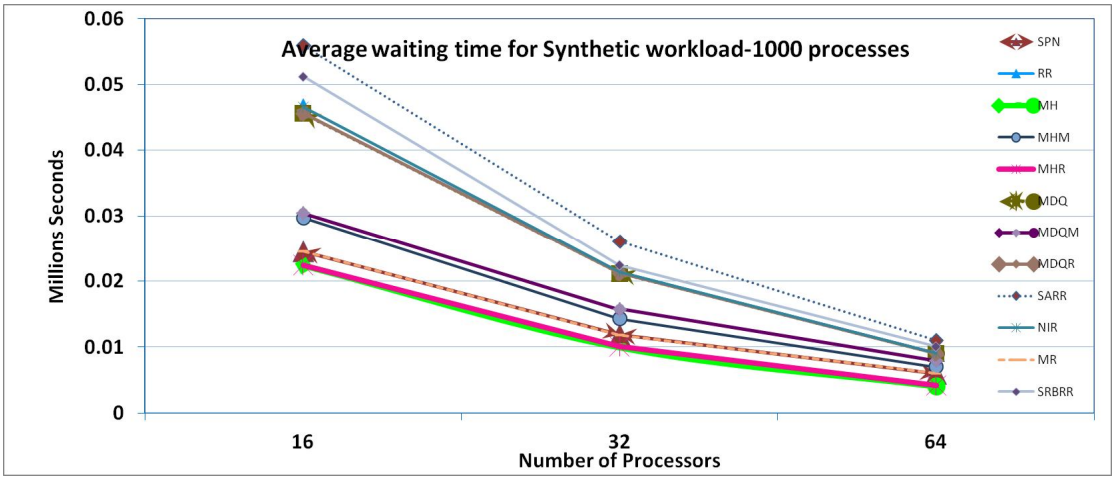

Figure 6.2(b): Average waiting times of twelve algorithms for synthetic workload of 1000 processes

Figure 6.2 (a, b) shows the average waiting time computed by each scheduling algorithm for synthetic workload trace of '1000' processes under the increased number of processors successively from '16' to '64'. Results from the above figures show that MH, MHR have shown the shortest average waiting times as compared to other algorithms. In addition, the average waiting times computed by SPN and MR scheduling algorithms are slightly higher than those values for MH and MHR. Figure 6.2(a, b) also presents that MHM, MDQM, MDQR, MDQ, RR, NIR, SARR and SRBRR have produced the higher average waiting time measures than MH, MHR, SPN and MR. Moreover, the PLRR, P, NRR, FCFS and LJF have shown the worst performance w. r. t. waiting times.

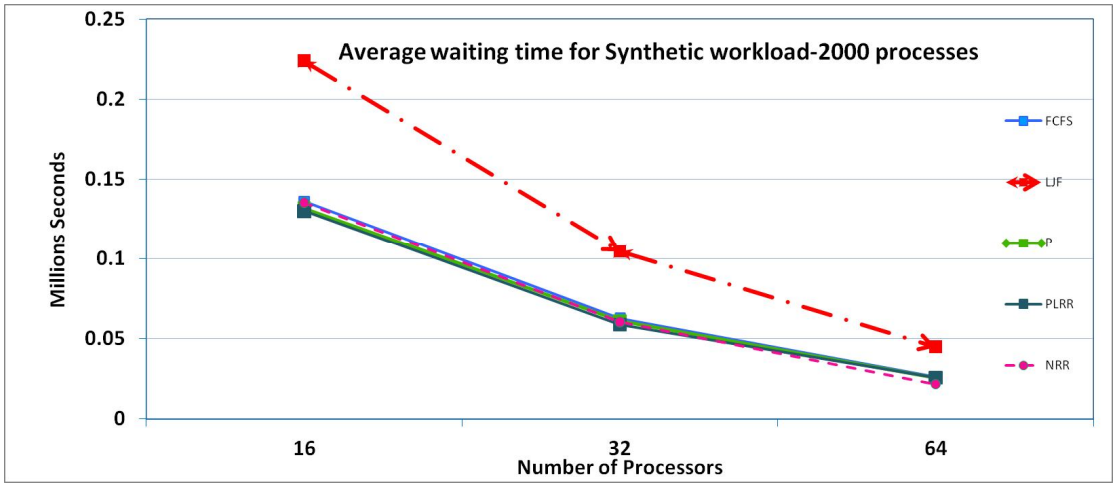

Figure 6.2(c): Average waiting time of five algorithms for synthetic workload of 2000 processes

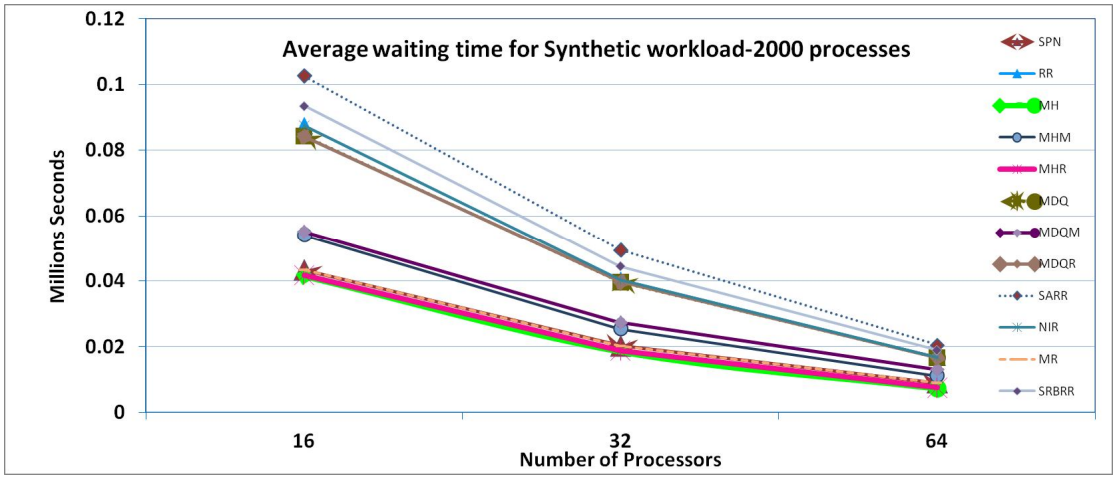

Figure 6.2(d): Average waiting time of twelve algorithms for synthetic workload of 2000 processes

Figure 6.2(c, d) shows the average waiting times for each scheduling algorithm for synthetic workload of '2000' processes. Each scheduling algorithm has shown the same pattern in representing average waiting times in Figure (c, d) as observed in Figure 6.2(a, b). All scheduling algorithms have shown the relative average waiting measures under the increased synthetic workload from '1000' processes to '2000' processes. Moreover, all scheduling algorithms also have shown the improvement w. r. t. average waiting times by varying the number of CPUs from '16' to 64'. As a result, proposed MH and MHR have shown the best average waiting time measures, relative performance, and support true scalability, under the increased synthetic workload and number of processors, in dynamic scheduling environment.

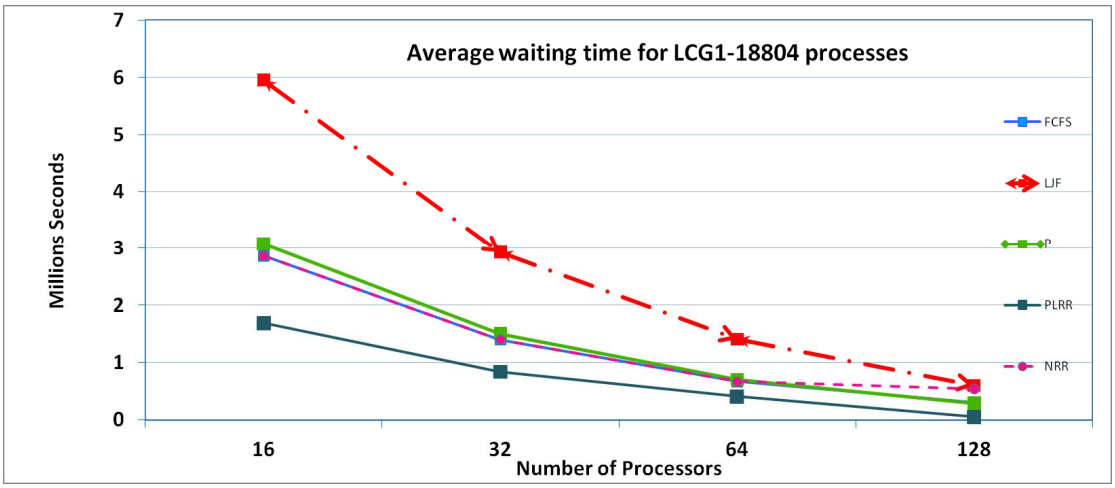

Figure 6.3(a): Average waiting time of five algorithms for 10% workload of LCG1

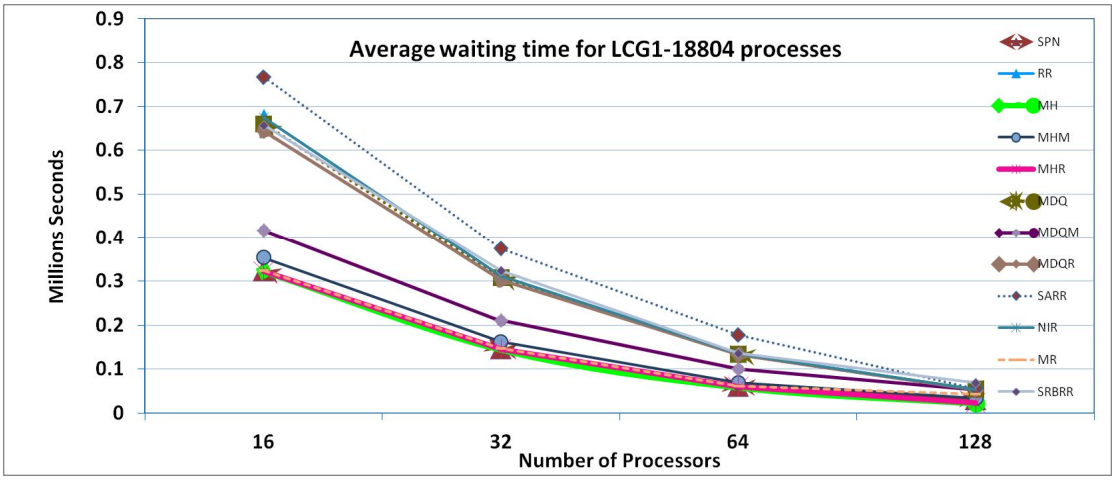

Figure 6.3(b): Average waiting times of twelve algorithms for 10% workload of LCG1

Figure 6.3(a, b) illustrates the average waiting times for scheduling algorithm using LCG1 workload trace of '18804' processes. It has been found that MH, MHR, SPN and MR scheduling algorithms have shown the best performance while producing the shortest average waiting times as compared to other scheduling algorithms. It also presents that MHM, MDQM, MDQR, MDQ, RR, NIR, SARR and SRBRR are at par in performance but result in higher average waiting time measures as compared to those for MH, MHR, SPN and MR. Moreover, the PLRR, P, NRR, FCFS and LJF have shown the worst performance w. r. t. the average waiting time measures. The LJF has shown to have the longest average waiting times. All scheduling algorithms have shown the improvement w. r. t. average waiting times by varying number of CPUs successively from '16' to '128'. As a result, MH and MHR have shown the best average waiting times for '10%' workload of LCG1.

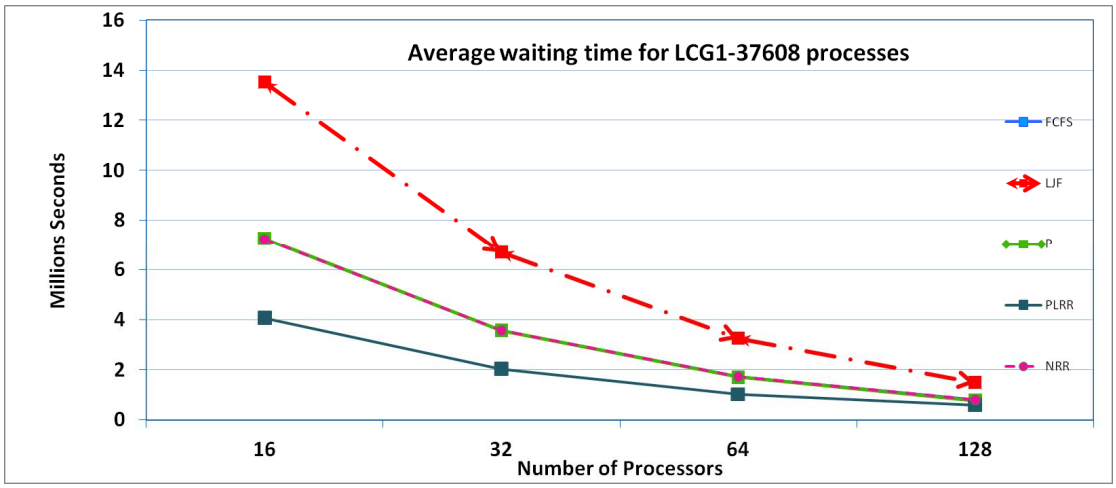

Figure 6.3(c): Average waiting times of five algorithms for 20% workload of LCG1

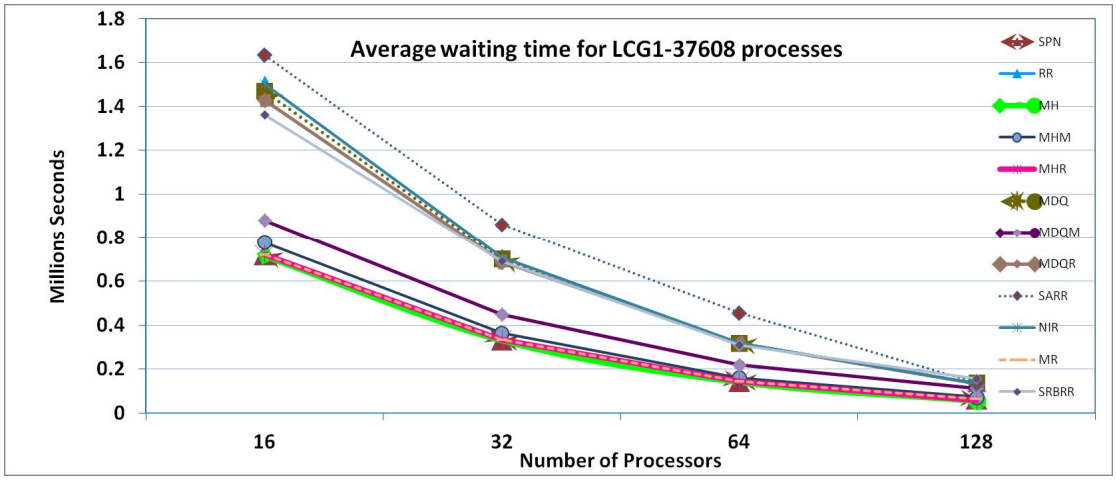

Figure 6.3(d): Average waiting times of twelve algorithms for 20% workload of LCG1

Figure 6.3(c, d) shows the average waiting times for scheduling algorithm using LCG1 workload trace of '37608' processes. It has been found that all scheduling algorithms have shown the same pattern for average waiting time measures, as obtained in Figure 6.3(a, b). All scheduling algorithm have shown the relative results w. r. t. average waiting time measures by increasing the workload from '10%' to '20%' of LCG1 workload.

Finally, all scheduling algorithms show scalability, as they maintain performance by increasing the workload and by varying the number of CPUs. Out of all scheduling algorithms, MH and MHR have shown the best average waiting times for both workload traces of LCG1.

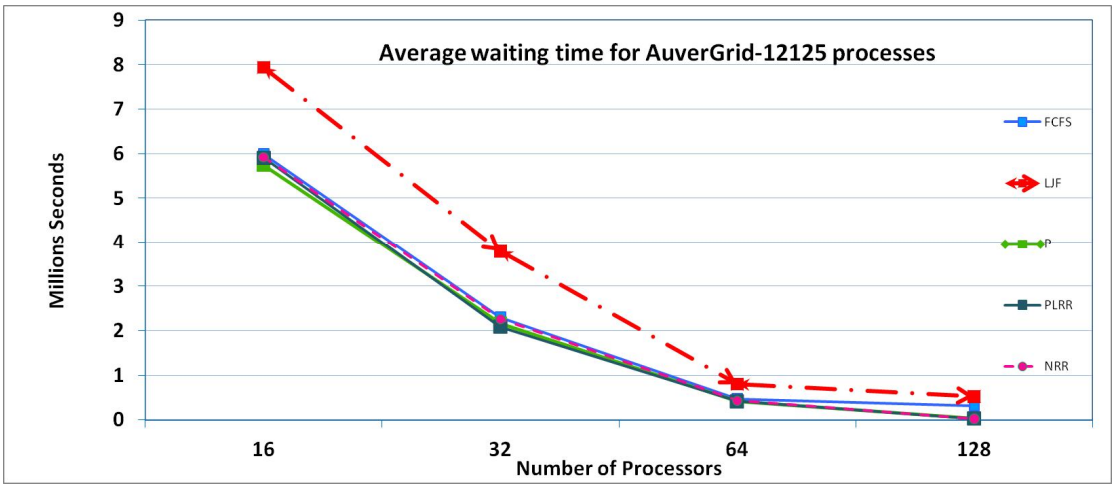

Figure 6.4(a): Average waiting times of five algorithms for 3% workload of AuverGrid

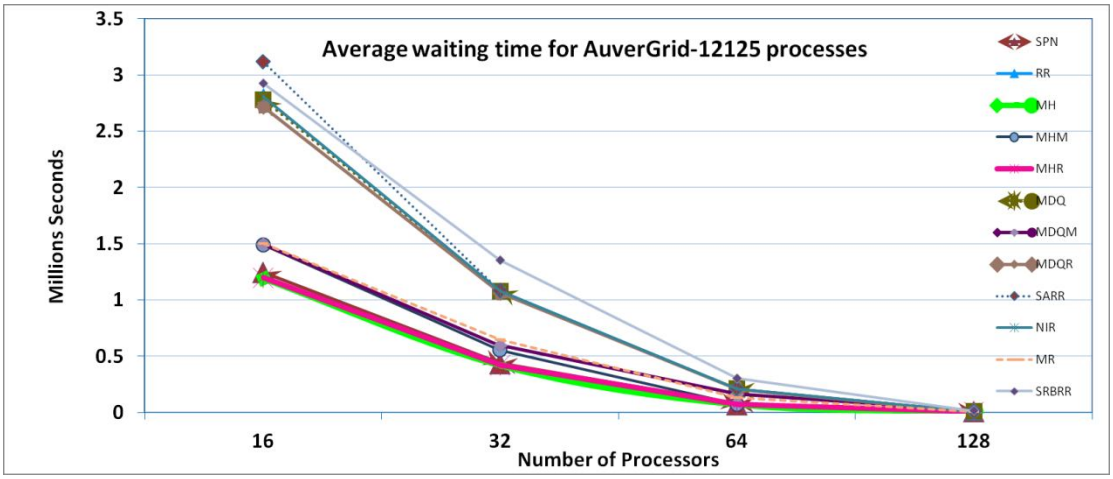

Figure 6.4(b): Average waiting times of twelve algorithms for 3% workload of AuverGrid

Figure 6.4 (a, b) shows the average waiting times computed for each scheduling algorithm using AuverGrid workload trace of '12125' processes using '16' to '128' processors. This figure portrays that MH, MHR, SPN and MR scheduling algorithms have shown the best performance in resulting the shortest average waiting times compared to other scheduling algorithms. Figure 6.4 (a, b) also shows that MHM, MDQM, MDQR, MDQ, RR, NIR, SARR and SRBRR are at same performance level, but result in higher average waiting time measures as compared to those for MH, MHR, SPN and MR.

Moreover, the PLRR, P, NRR, FCFS and LJF have shown the worst performance w. r. t. the average waiting time measures. The LJF has shown to have the longest average waiting times.

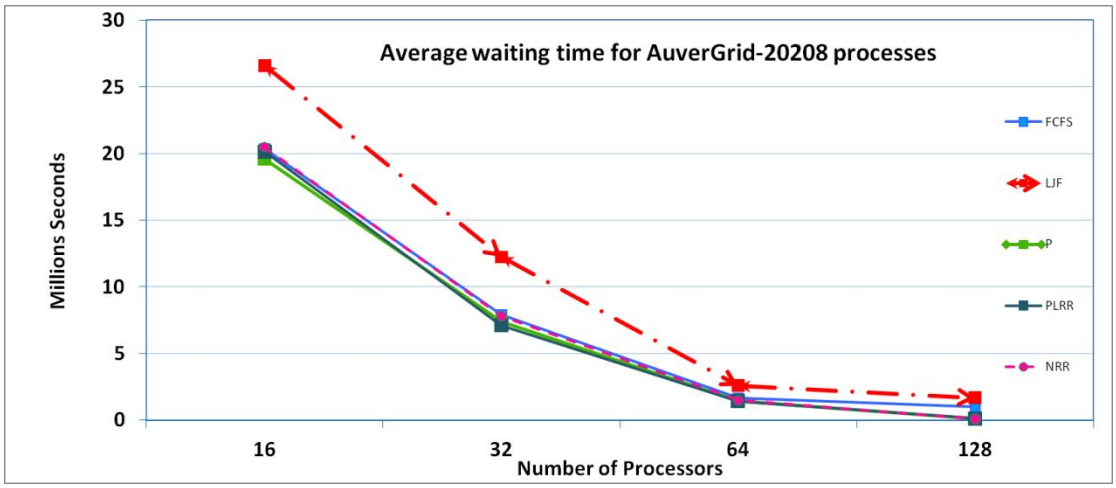

Figure 6.4(c): Average waiting times of five algorithms for 5% workload of AuverGrid

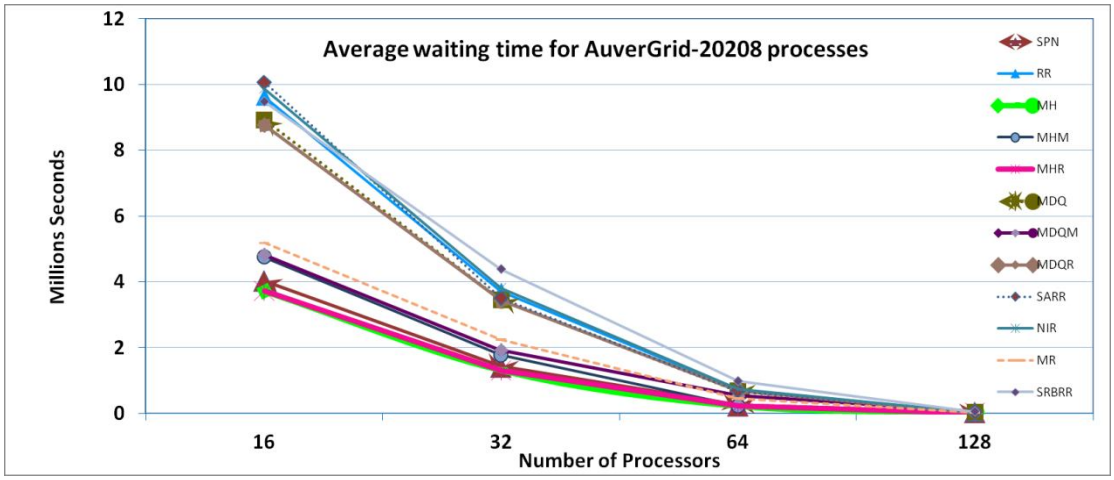

Figure 6.4(d): Average waiting times of twelve algorithms for 5% workload of AuverGrid

Figure 6.4 (c, d) shows the average waiting time computed for each scheduling algorithm using AuverGrid workload trace of '20208' processes and also has shown the same performance pattern, as obtained in Figure 6.4(a, b). All scheduling algorithms have shown scalability under increasing workload of AuverGrid from '3%' to '5%' and by varying the number of processors from '16' to '128'. In addition, the same performance patterns have been observed in Figure 6.4(a, b, c, d) for AuverGrid workload traces as found in Figure 6.2(a, b, c, d) and Figure 6.3(a, b, c, d) when analyzed for synthetic and LCG1 workload traces. However, from Figure 6.2(a, b, c, d) - Figure 6.4(a, b, c, d), it is observed that the obtained values of average waiting times for each algorithm is independent of the type of workload, size of workload and the number of CPUs used for computation. The MH and MHR have shown the best average waiting times for all types of traces under dynamic grid environment.

6.6.4.2 Average Turnaround Time Analysis

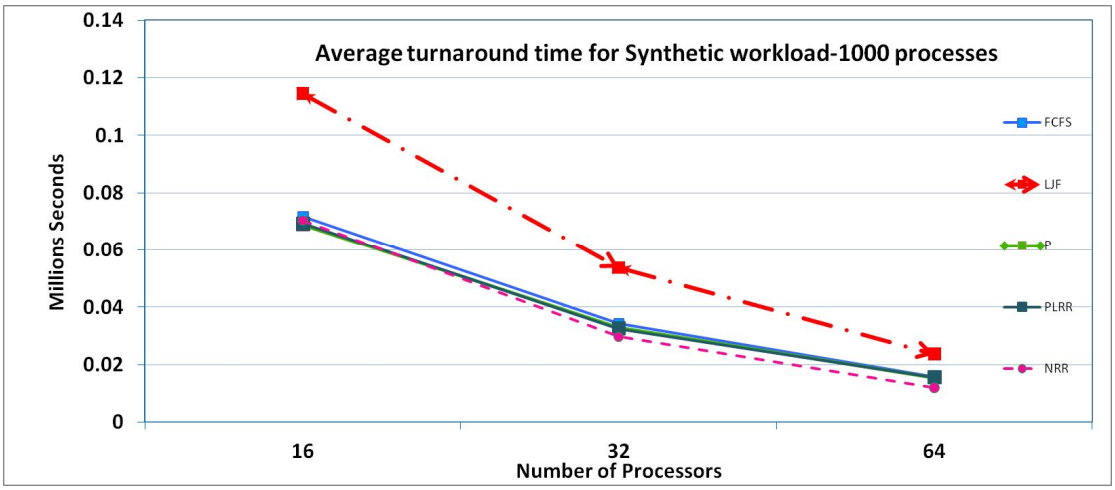

Figure 6.5(a): Average turnaround times of five algorithms for synthetic workload of 1000 processes

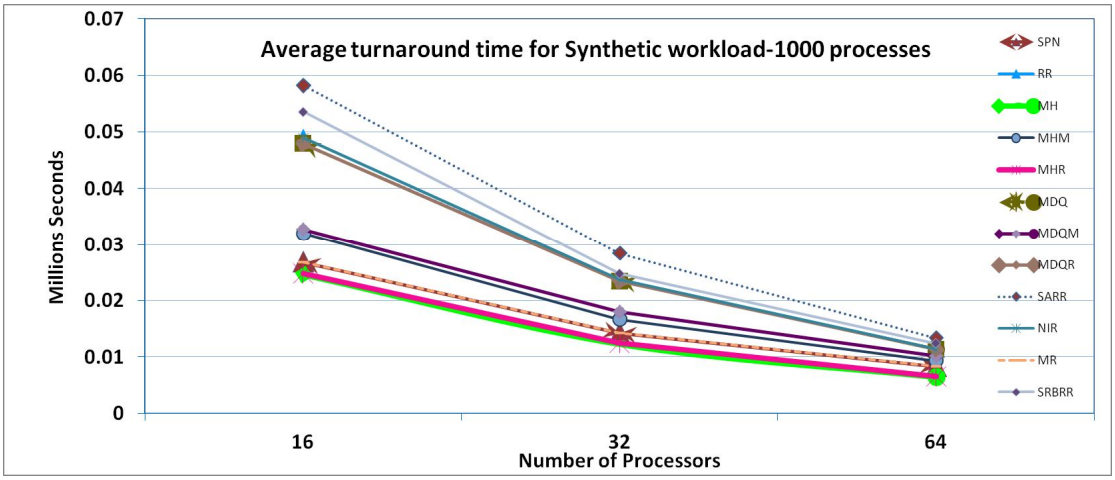

Figure 6.5(b): Average turnaround times of twelve algorithms for synthetic workload of 1000 processes

Figure 6.5 (a, b) shows that MH and MHR produces the shortest average turnaround times compared to all other scheduling algorithms for synthetic workload. It can also be interpreted that SPN, MHM and MR show better performance w. r. t. average turnaround times but higher than the values those for MH and MHR. It is observed that MDQM, MDQR, MDQ, RR, NIR, SARR and SRBRR have shown the similar type of results w. r. t. the average turnaround time measures. However, it is found that P, NRR, FCFS and LJF show the worst performance, out of which LJF has the longest average turnaround times. Moreover, all scheduling algorithms have shown the improvement w. r. t. average turnaround time measures under increasing the number of CPUs successively from '16' to '64'.

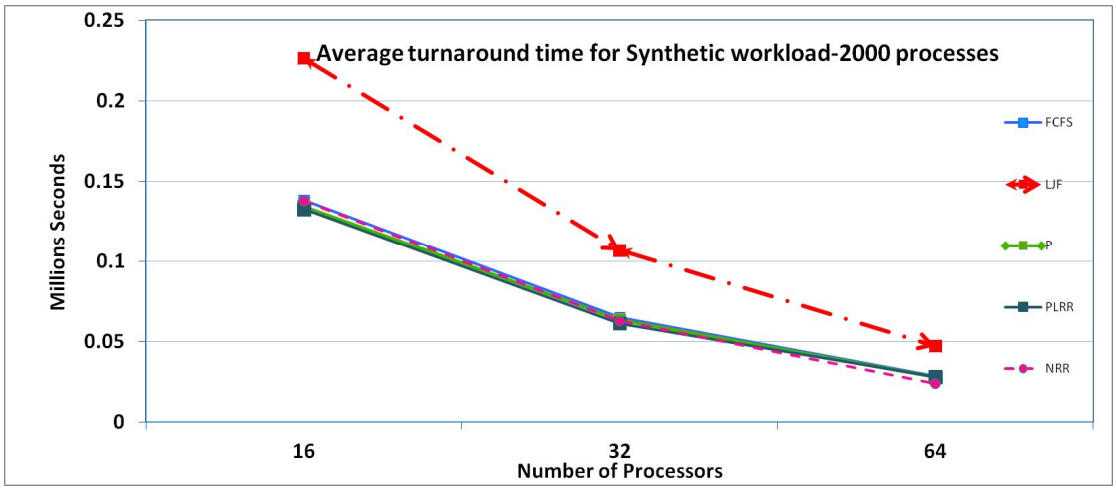

Figure 6.5(c): Average turnaround times of five algorithms for synthetic workload of 2000 processes

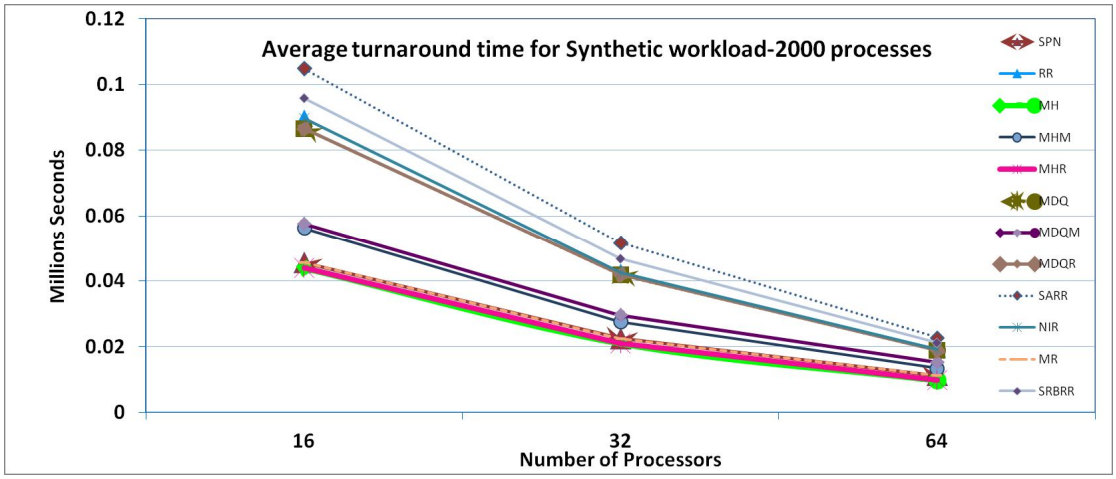

Figure 6.5(d): Average turnaround times of twelve algorithms for synthetic workload of 2000 processes

Figure 6.5 (c, d) shows the average turnaround times computed for each scheduling algorithm using synthetic workload trace of '2000' processes. It can be observed that all scheduling algorithms have shown relative measures of average turnaround times, as observed and analyzed from Figure 6.5 (a, b). Figure 6.5 (a, b, c, d) shows that all scheduling algorithms have shown the relative performance under the increasing workload. It also shows that all scheduling algorithms have shown improvement w. r. t. average turnaround times by increasing the number of CPUs successively from '16' to '64'. Hence, all scheduling algorithms have shown scalability in dynamic scheduling environment. As a result, MH and MHR have shown the best average turnaround times for synthetic workload traces.

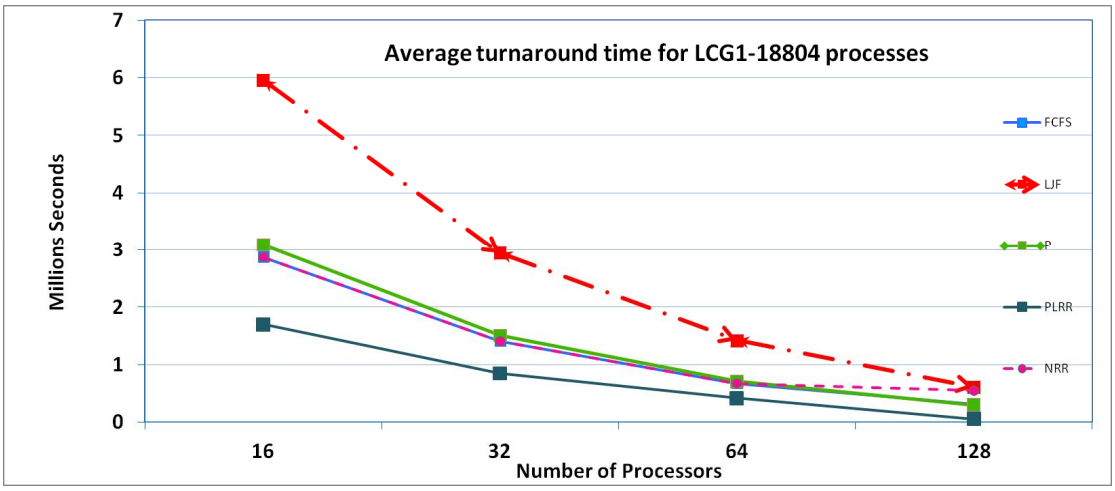

Figure 6.6(a): Average turnaround times of five algorithms for 10% workload of LCG1

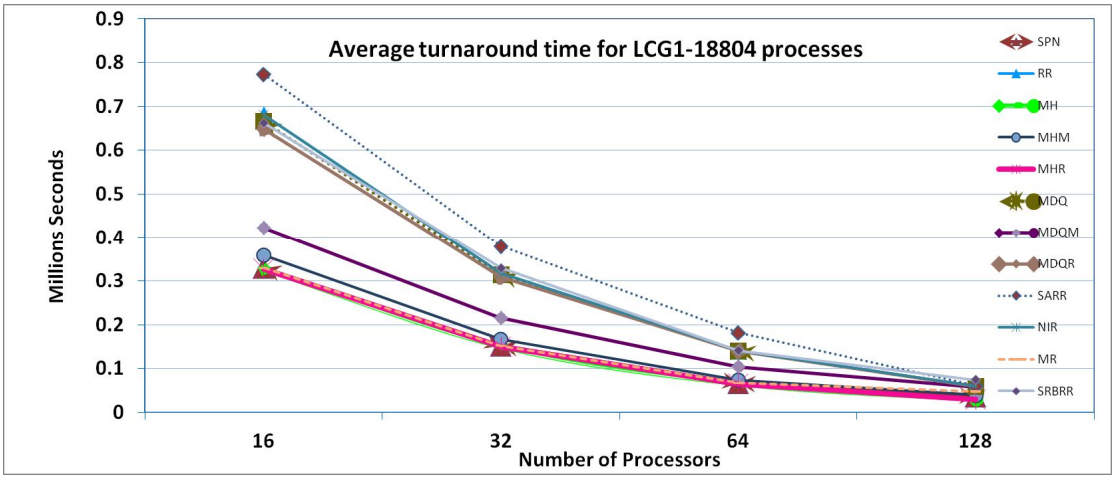

Figure 6.6(b): Average turnaround times of twelve algorithms for 10% workload of LCG1

Figure 6.6(a, b) shows the average turnaround times computed for each scheduling algorithm using '10%' workload of LCG1. The values for average turnaround times computed by MH and MHR are found shorter than those for the other grid scheduling algorithms. This figure also shows that SPN, MHM and MR have shown better performance w. r. t. the average turnaround times. With respect to the average turnaround time's measures, MDQM, MDQR, MDQ, RR, NIR, SARR and SRBRR have shown similar performance.

Furthermore, it is found that P, NRR, FCFS and LJF scheduling algorithms have the worst performances, which result in longer turnaround times, out of which LJF has shown the longest average turnaround time measures.

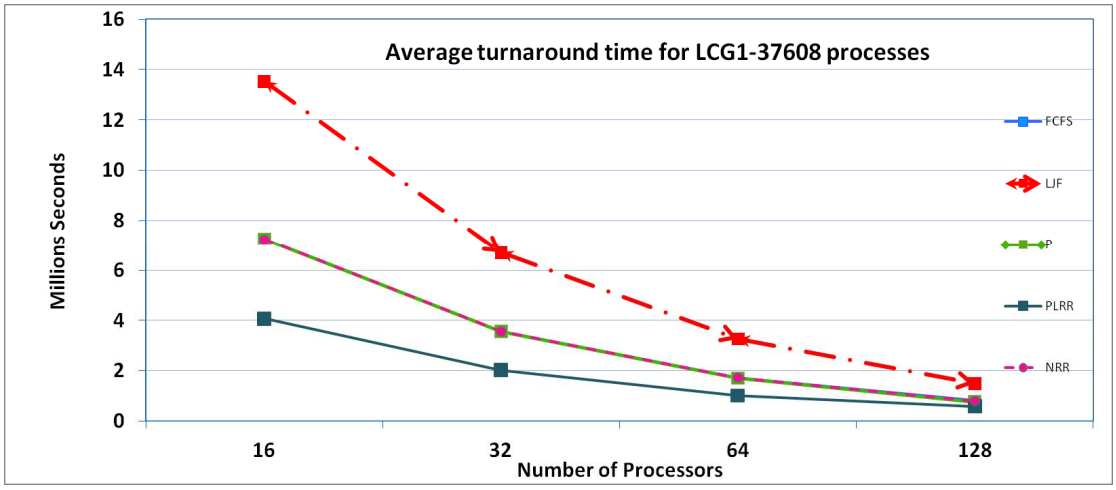

Figure 6.6(c): Average turnaround times of five algorithms for 20% workload of LCG1

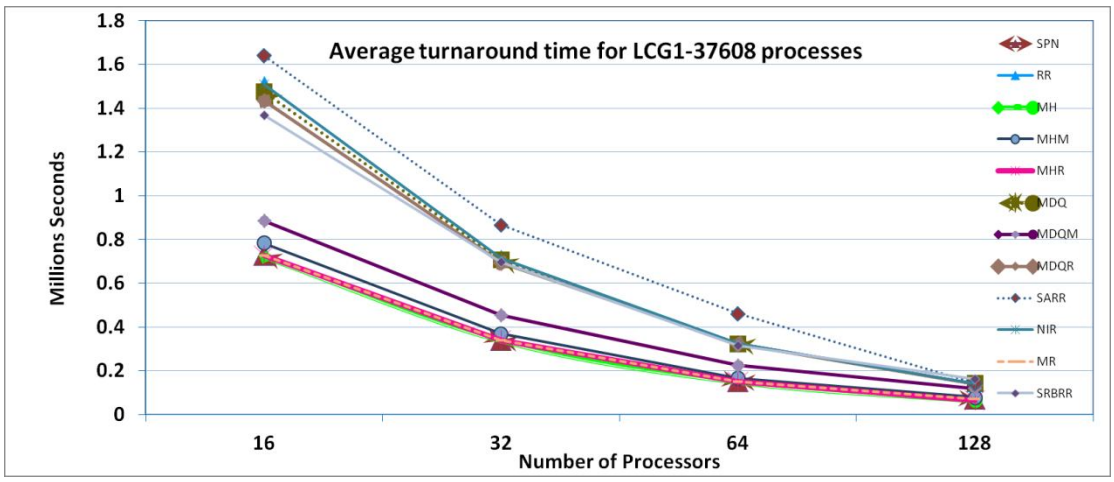

Figure 6.6(d): Average turnaround times of twelve algorithms for 20% workload of LCG1

Figure 6.6(c, d) shows the average turnaround times computed for each scheduling algorithm using '20%' workload of LCG1. Results have shown the same pattern, as seen and analyzed from Figure 6.6(a, b) obtained for '10%' workload of LCG1. It is observed that all scheduling algorithms have shown the relative performance measures of average turnaround times under the increasing workload of LCG1 from '10%' to '20%'. Moreover, all scheduling algorithms have shown the reduced values of average turnaround times under increasing the number of CPUs successively from '16' to '128'. All scheduling algorithms have shown support for scalability under the increased workload and varied number of CPUs in dynamic scheduling environment. Out of seventeen scheduling algorithms, MH and MHR have shown the best average turnaround times.

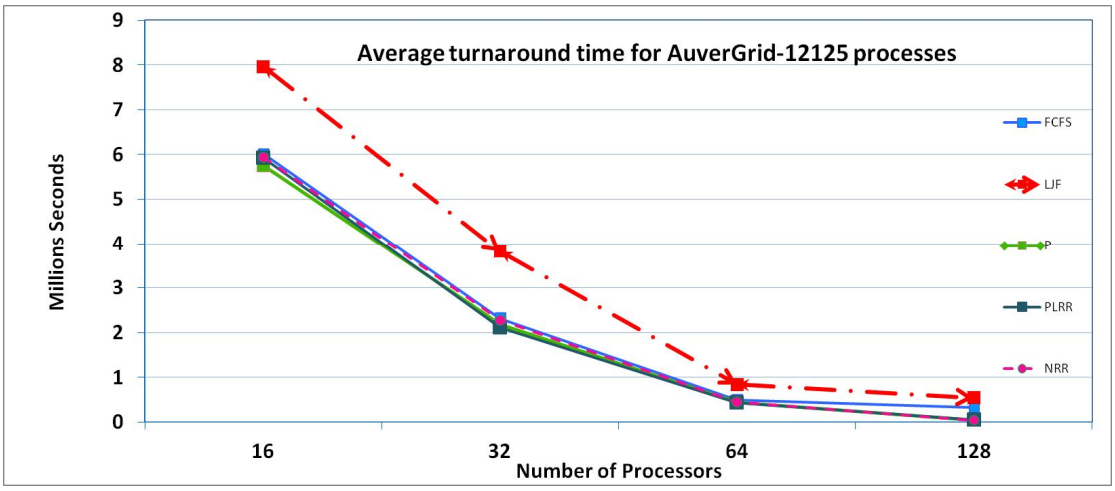

Figure 6.7(a): Average turnaround times of five algorithms for 3% workload of AuverGrid

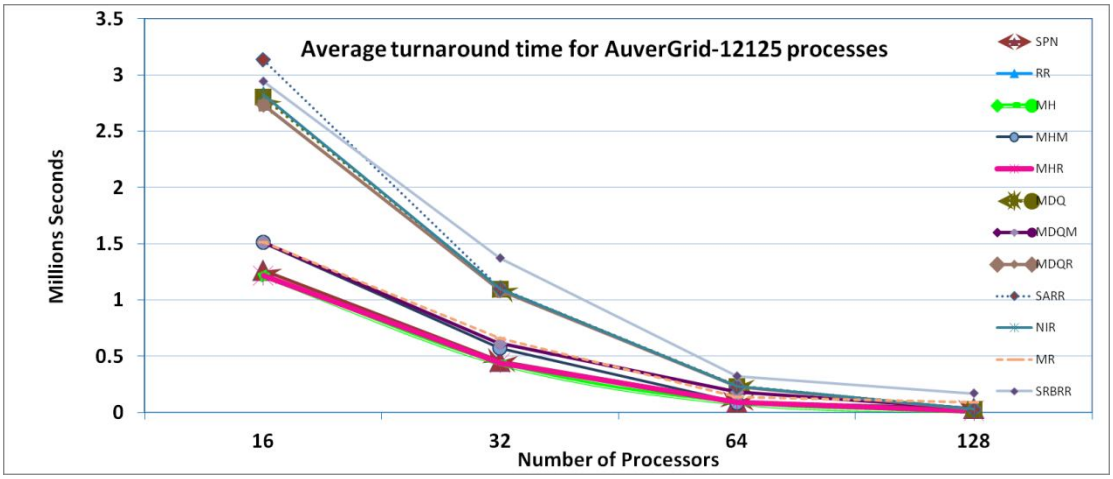

Figure 6.7(b): Average turnaround times of twelve algorithms for 3% workload of AuverGrid

The average turnaround times computed for each scheduling algorithm using AuverGrid workload trace of '12125' processes are shown in Figure 6.7(a, b). MH and MHR have shown the shorter average turnaround times compared to the other grid scheduling algorithms. SPN, MHM and MR have shown better average waiting time measures but longer than those for MH and MHR. Figure 6.7(a, b) also shows that SPN, MHM and MR have better performance w. r. t. the average turnaround times. In order to show the average performance w. r. t. the average turnaround times, MDQM, MDQR, MDQ, RR, NIR, SARR and SRBRR are found most suiTable candidates. Furthermore, it is found that P, NRR, FCFS, PLRR and LJF scheduling algorithms have shown the worst performance. All scheduling algorithms have shown improved average turnaround time measures by increasing CPUs.

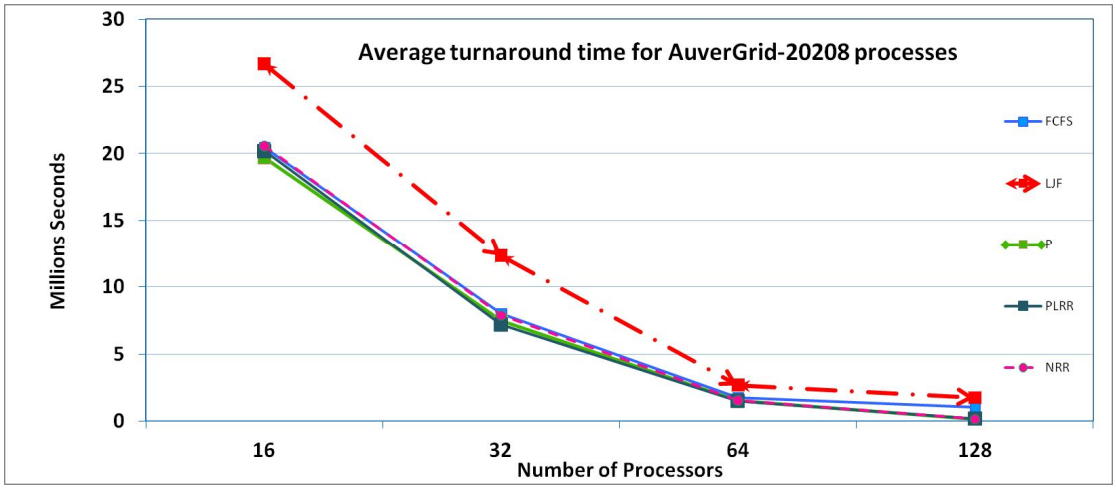

Figure 6.7(c): Average turnaround times of five algorithms for 5% workload of AuverGrid

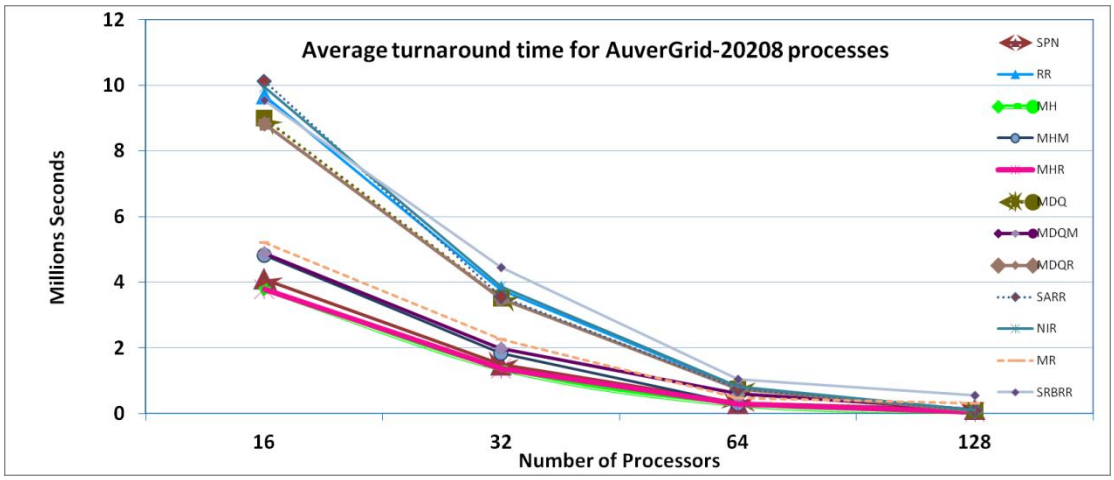

Figure 6.7(d): Average turnaround times of twelve algorithms for 5% workload of AuverGrid

Figure 6.7(c, d) shows the average turnaround time for each scheduling algorithm for '5%' workload of AuverGrid. Each scheduling algorithm has shown relative performance under the increased workload from '3%' to '5%' of AuverGrid workload. Moreover, Figure 6.7(a, b, c, d) has shown the same performance pattern w. r. t. average turnaround time measures for each scheduling algorithm, as observed in Figure 6.5(a, b, c, d) and Figure 6.6(a, b, c, d) for synthetic and LCG1 workload traces respectively. Finally, Figure 6.5(a, b, c, d)- Figure 6.7(a, b, c, d) shows that all scheduling algorithms posses that relative performance w. r. t. average turnaround time measures independent of the type of the workload, the workload size and the number of CPUs used in the experimentation. It is also found that each algorithm supports true scalability under the increased workload and varied number of CPUs in the dynamic grid scheduling environment. In summary, MH and MHR have shown best average turnaround times.

6.6.4.3 Average Response Time Analysis

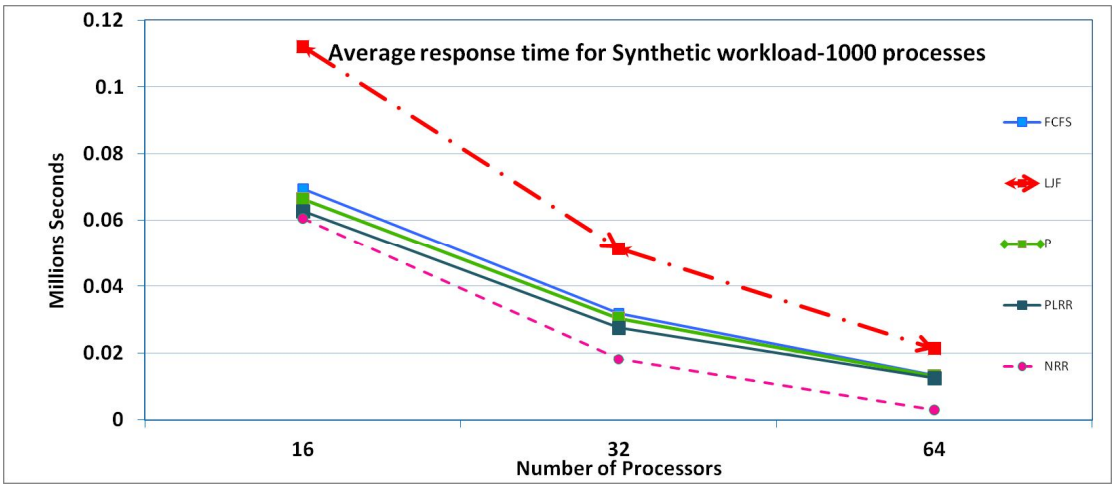

Figure 6.8(a): Average response times of five algorithms for synthetic workload of 1000 processes

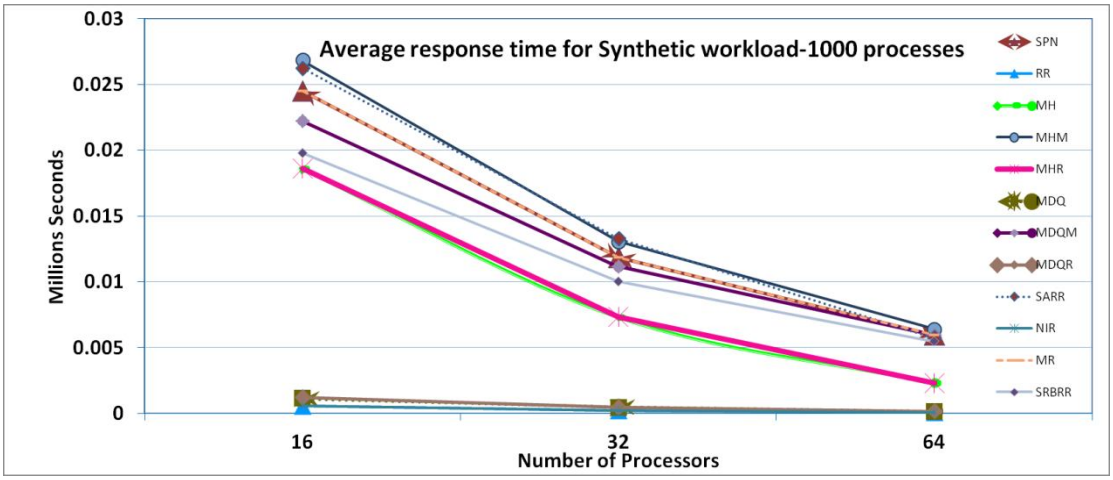

Figure 6.8(b): Average response times of twelve algorithms for synthetic workload of 1000 processes

Figure 6.8(a, b) shows the average response times computed for the scheduling algorithms using synthetic workload traces of '1000' processes. It is clear from the figure that average response times computed by the RR, NIR, MDQ and SARR are shorter than other scheduling algorithms. In addition, MDQR, SRBRR, MDQM, MH and MHR are found to be at good level w. r. t. the average response times. Moreover, the SPN, MR, MHM have shown the longer response times whilst PLRR, P, NRR, FCFS and LJF have shown the worst performance w. r. t. average response time measures. In addition, LJF is found to have the longest average response times. Finally, Figure 6.8(a, b) shows that all scheduling algorithms have shown improvement w. r. t. the response time measures by increasing the number of CPUs successively from '16' to '64'.

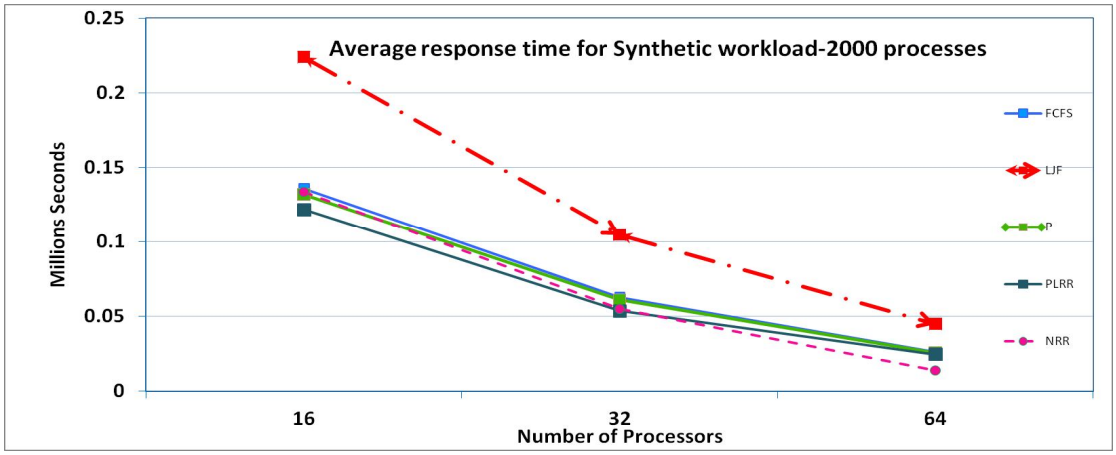

Figure 6.8(c): Average response times of five algorithms for synthetic workload of 2000 processes

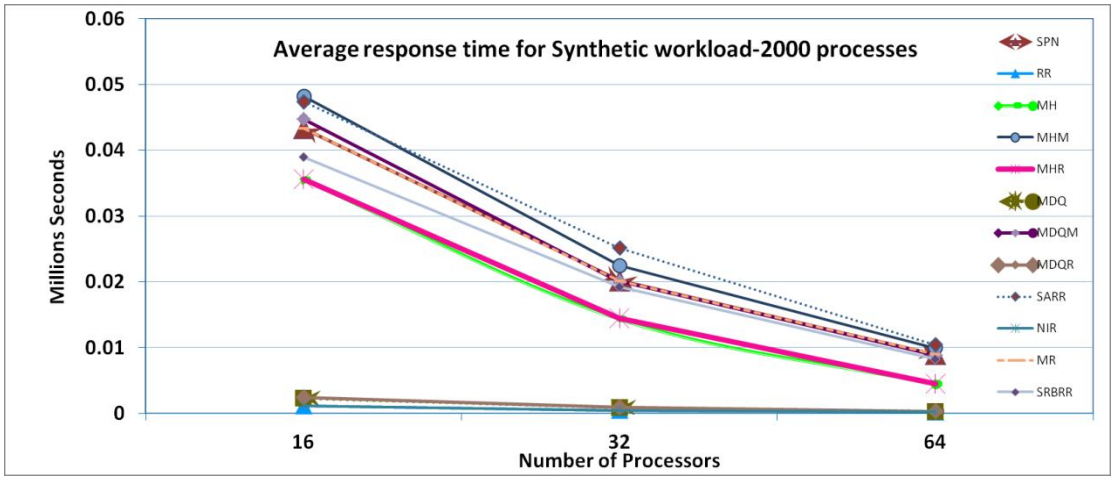

Figure 6.8(d): Average response times of twelve algorithms for synthetic workload of 2000 processes

Figure 6.8(c, d) shows the average response times computed for the scheduling algorithms using synthetic workload traces of '2000' processes. It can be seen from the figures that average response times computed by the scheduling algorithms have shown the same pattern, as observed and analyzed in Figure 6.8(a, b) for synthetic workload trace of '1000' processes. Figure 6.8(a, b, c, d) shows that all scheduling algorithms have shown the improvement w. r. t. the average response time measures by increasing the number of CPUs successively from '16' to '64'. Finally, all scheduling algorithms support scalability while maintaining the performance under the increased synthetic workload from '1000' to '2000' processes. Moreover, all scheduling algorithms have shown improvement w. r. t. the response times under the increased number of CPUs. As a result, RR, NIR, MDQ and SARR have shown shorter average response times than other scheduling algorithms.

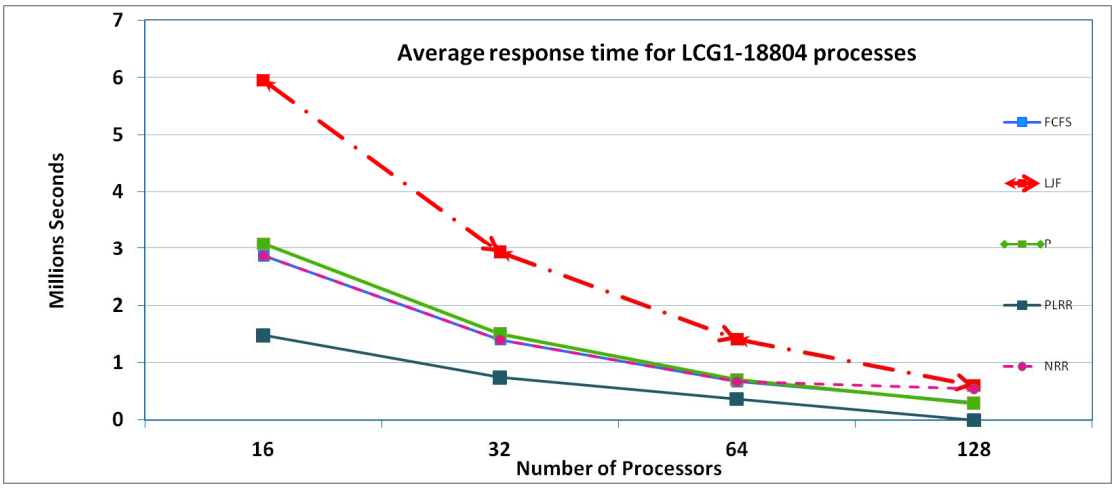

Figure 6.9(a): Average response times of five algorithms for 10% workload of LCG1

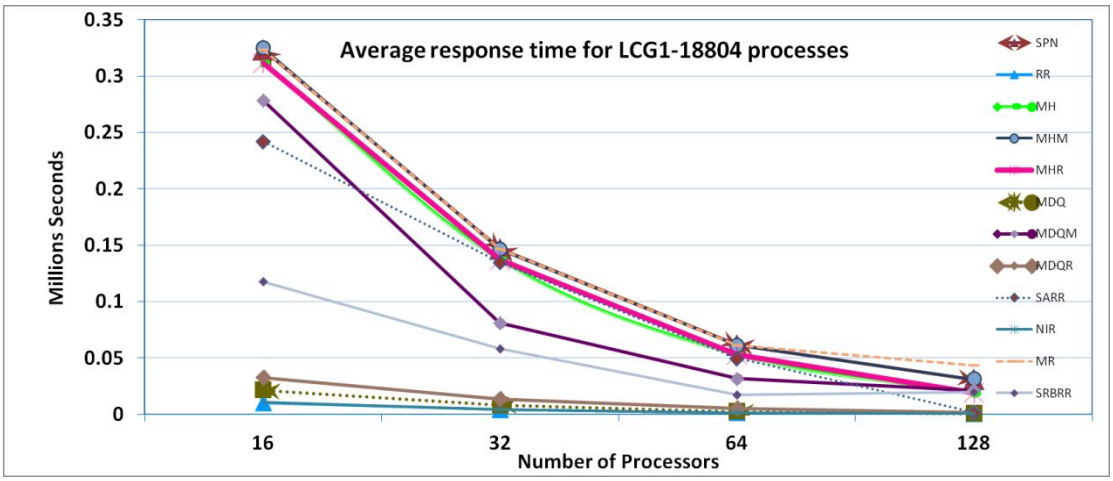

Figure 6.9(b): Average response times of twelve algorithms for 10% workload of LCG1

Average response times computed for the scheduling algorithms using '10%' workload of LCG1, are shown in Figure 6.9(a, b). It is found that that average response times computed by the RR, NIR, MDQ and SARR are shorter than other scheduling algorithms. Average response times for each algorithm have decreased by increasing the number of CPUs. It also shows that MDQR, SRBRR, MDQM, MH and MHR algorithms produces better average response time compared to other algorithms. However, the SPN, MR, MHM, PLRR have shown the longer response times whilst P , NRR, FCFS and LJF have shown the worst performance w. r. t. average response time measures, out of which LJF results in the longest average response times. All scheduling algorithms have shown the improvement w. r. t. average response time measures by increasing the number of CPUs successively from '16' to '128'.

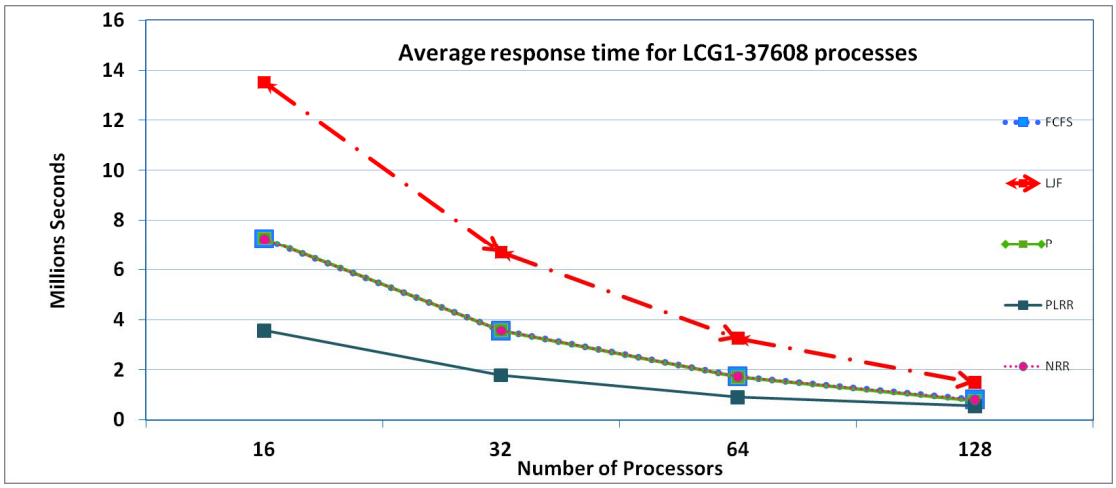

Figure 6.9(c): Average response times of five algorithms for 20% workload of LCG1

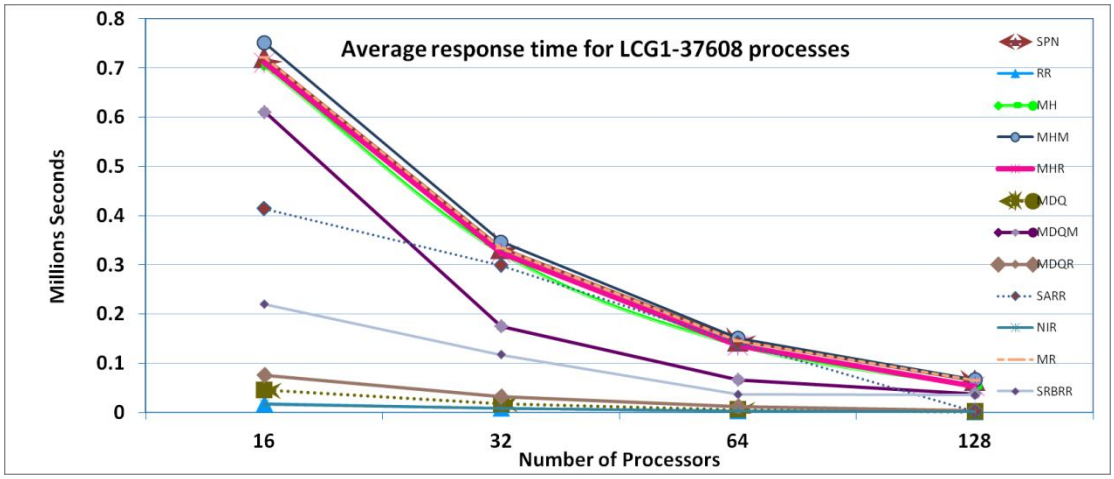

Figure 6.9(d): Average response times of twelve algorithms for 20% workload of LCG1

Figure 6.9(c, d) shows the average response times computed for the scheduling algorithms using '20%' workload of LCG1. From the Figure 6.9(a, b), it is clear that, performance pattern is more or less same w. r. t. average response measures for each scheduling algorithm, using '10%' workload of LCG1.

By increasing the workload of LCG1 from '10%' to '20%', all scheduling algorithms have shown relative performance. Moreover, all scheduling algorithms have shown the scalability under the increasing workload of LCG1 and by varying the number of CPUs. Finally RR, NIR, MDQ and SARR have shown the best average response times compared to other scheduling algorithms for both workload traces of LCG1 under dynamic grid scheduling environment.

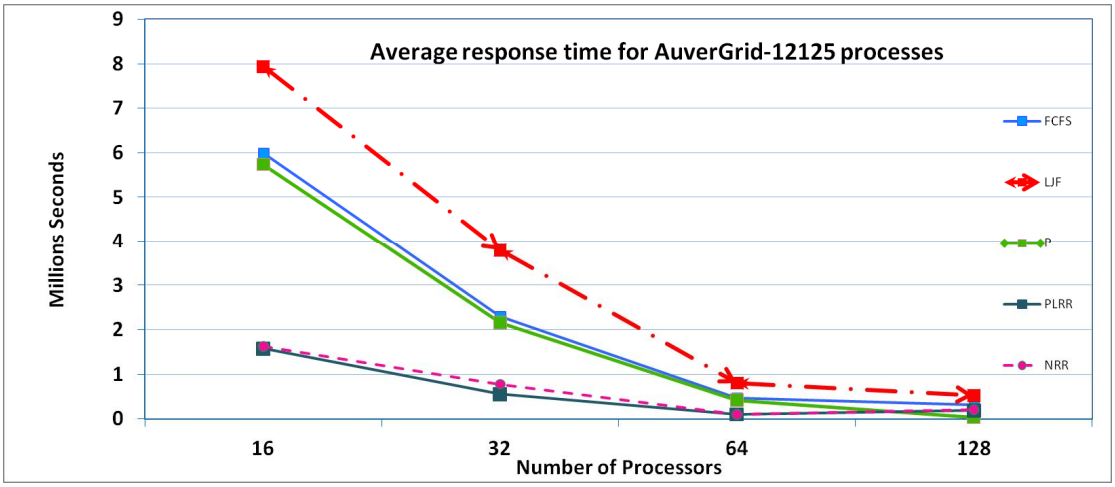

Figure 6.10(a): Average response times of five algorithms for 3% workload of AuverGrid

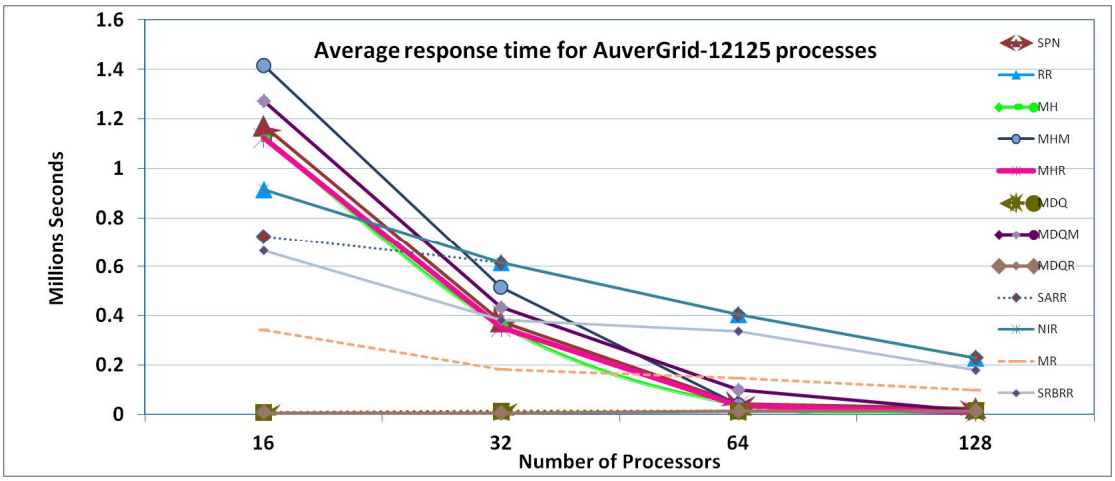

Figure 6.10(b): Average response times of twelve algorithms for 3% workload of AuverGrid

Figure 6.10(a, b) shows the average response times computed for each scheduling algorithm using AuverGrid workload trace of '12125' processes. It is found that average response times computed by the MDQ and MDQR are shorter than other scheduling algorithms. The average response times obtained by MR are slightly higher than the values for MDQ and MDQR. RR has shown the poor average response times compared to the values for MDQ, MDQR and MR as depicted in Figure 6.10(a, b). Average response times computed by SARR, SRBRR, MDQM, MH and MHR algorithms are higher than those for MDQ and MDQR; and have shown deviation in results in comparison with MR and RR. However, the SPN, NIR and MHM have shown the longer response times whilst P, PLRR, NRR, FCFS and LJF have shown the worst performance w. r. t. average response time measures. All scheduling algorithms have shown the improvement w. r. t. average response times by varying the number of CPUs successively from '16' to '128'.

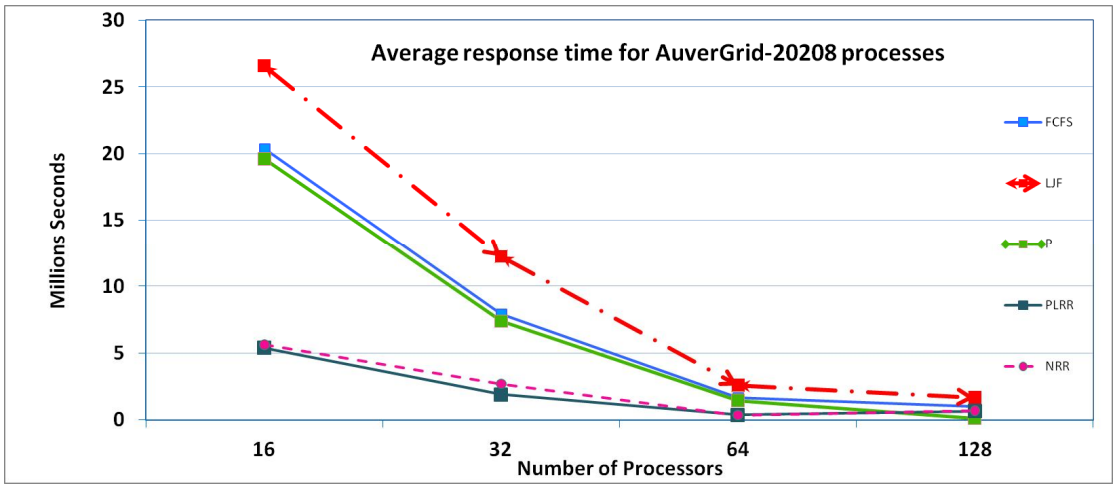

Figure 6.10(c): Average response times of five algorithms for 5% workload of AuverGrid

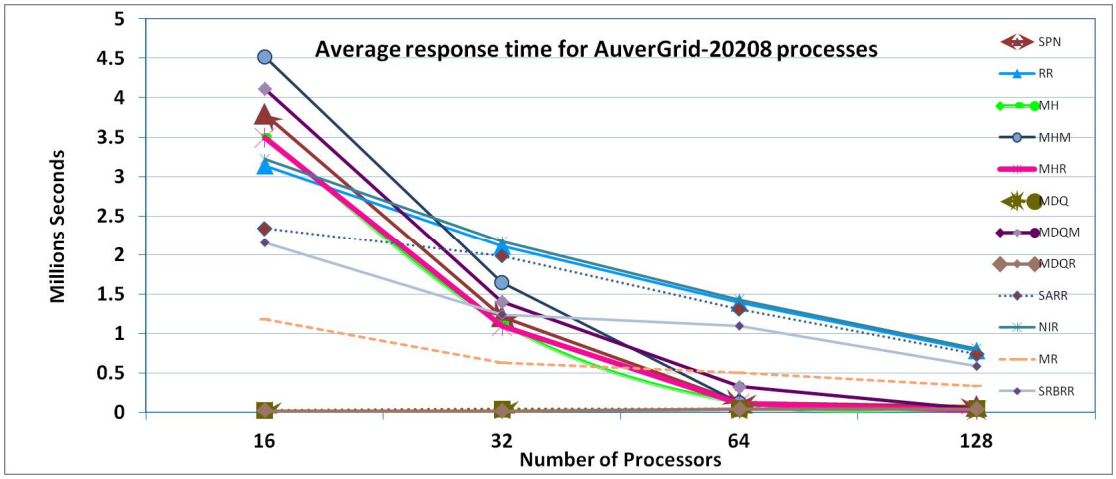

Figure 6.10(d): Average response times of twelve algorithms for 5% workload of AuverGrid

Figure 6.10(c, d) has shown the same performance pattern of average response times for each scheduling algorithm using '5%' workload of AuverGrid, as observed in Figure 6.10(a, b) for each algorithm using '3%' workload of AuverGrid. RR, NIR, MDQ and SARR have shown the best performance compared to other scheduling algorithms using synthetic and LCG1 workload traces under dynamic scheduling environment as depicted in Figure 6.8 (a, b, c, d) and Figure 6.9(a, b, c, d). RR has shown the poor average response times for AuverGrid workload traces as shown in Figure 6.10(a, b, c, d). The probable reason in this is, too many jobs were arrived in the queue within a very short time interval and queue size had become very large. In this situation, RR could not give better response to processes, that is why it has shown the poor average response times[163]. Finally, all scheduling algorithms, except RR, have shown relative average response times independent of type of the workload, workload size and number of CPUs.

6.6.4.4 Average Slowdown Time Analysis

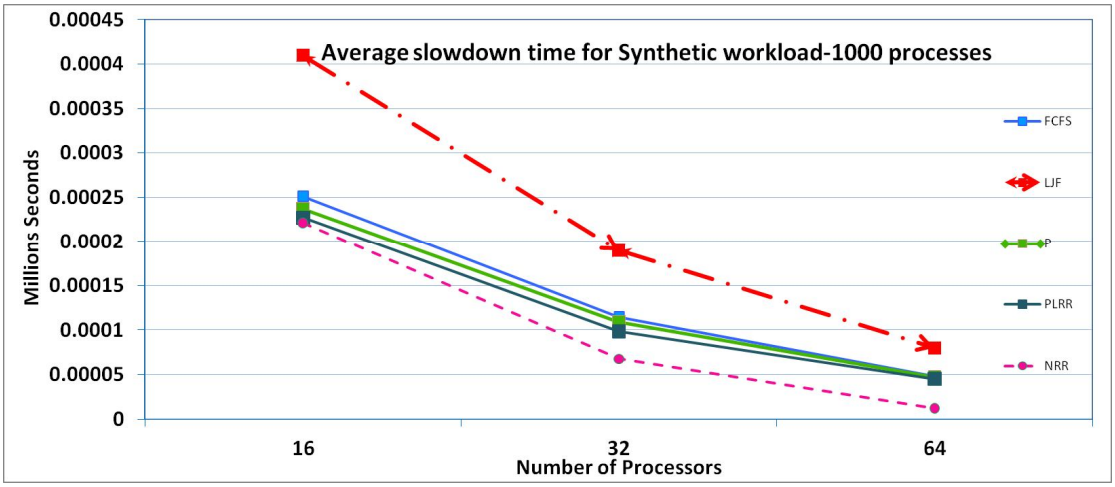

Figure 6.11(a): Average slowdown times of five algorithms for synthetic workload of 1000 processes

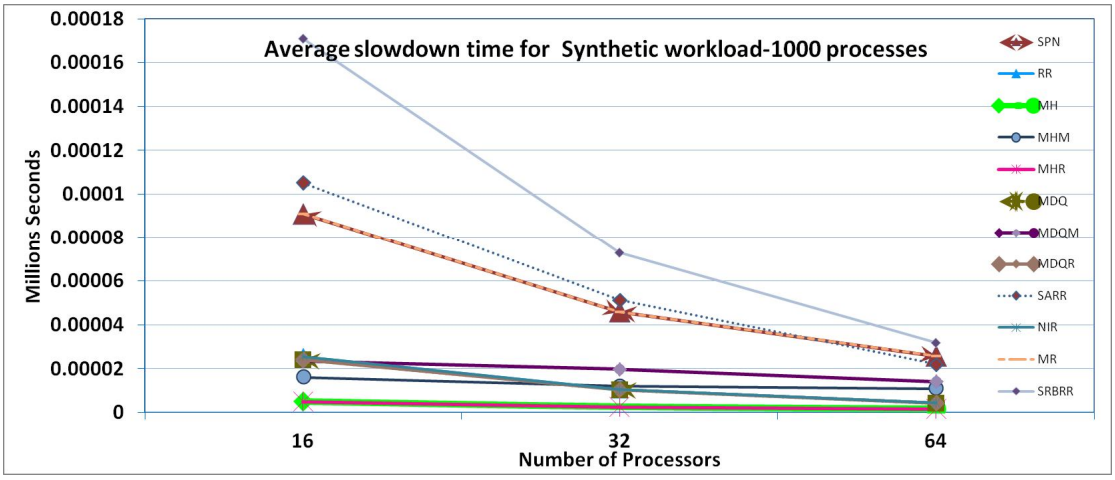

Figure 6.11(b): Average slowdown times of twelve algorithms for synthetic workload of 1000 processes

The average slowdown times computed for each scheduling algorithm using synthetic workload trace of '1000' processes are depicted in Figure 6.11(a, b). It is demonstrated that MH and MHR produces the shortest average slowdown time compared to other scheduling algorithms. MHM, RR, NIR, MDQ, MDQR and SARR have shown better average slowdown times but higher than those for MH and MHR. It is clear from the Figure 6.11(a, b) that MDQM, SPN, MR and SRBRR seems to have average performance w. r. t. the average slowdown times. Furthermore, the PLRR, P, NRR, FCFS and LJF have shown the worst performance w. r. t. average slowdown times, out of which LJF has resulted in the longest average slowdown times.

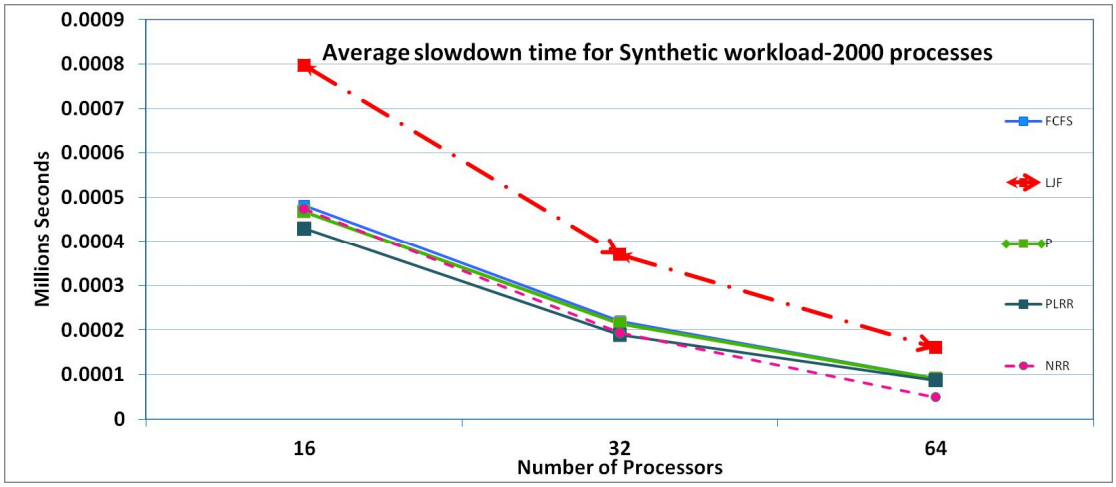

Figure 6.11(c): Average slowdown times of five algorithms for synthetic workload of 2000 processes

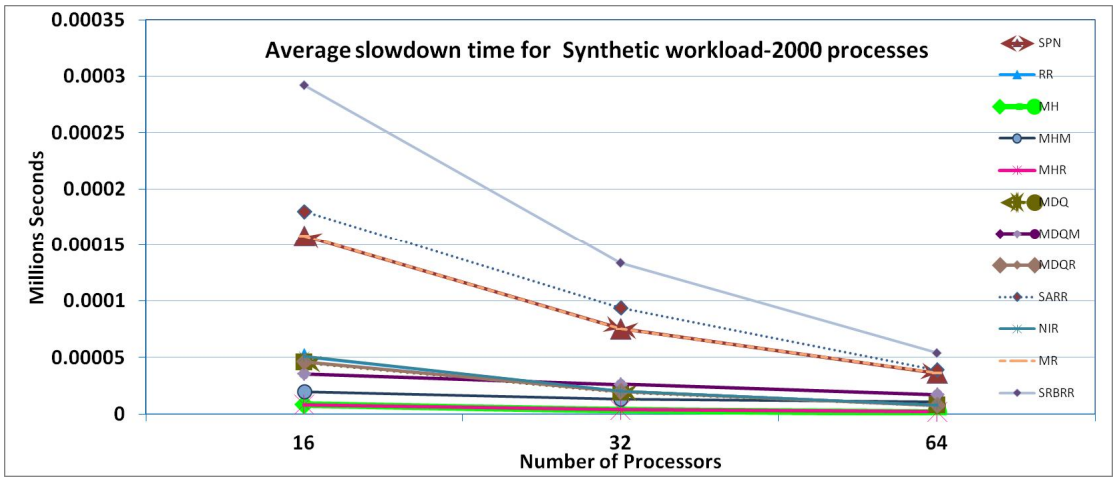

Figure 6.11(d): Average slowdown times of twelve algorithms for synthetic workload of 2000 processes

Figure 6.11(c, d) shows the average slowdown times computed for each scheduling algorithm using synthetic workload of '2000' processes. The performance of all scheduling algorithms have followed the same patterns w. r. t. average slowdown time measures as observed in Figure 6.11(a, b) for synthetic workload of '1000' processes. Figure 6.11(a, b, c, d) shows that all scheduling algorithms have maintained their performance under the increased synthetic workload from '1000' to '2000' processes. Moreover, all scheduling algorithms have shown improvement w. r. t. average slow down times under the increasing number of CPUs successively from '16' to '64'. Finally, MH and MHR have shown best average slowdown times for both synthetic workload traces under dynamic scheduling environment.

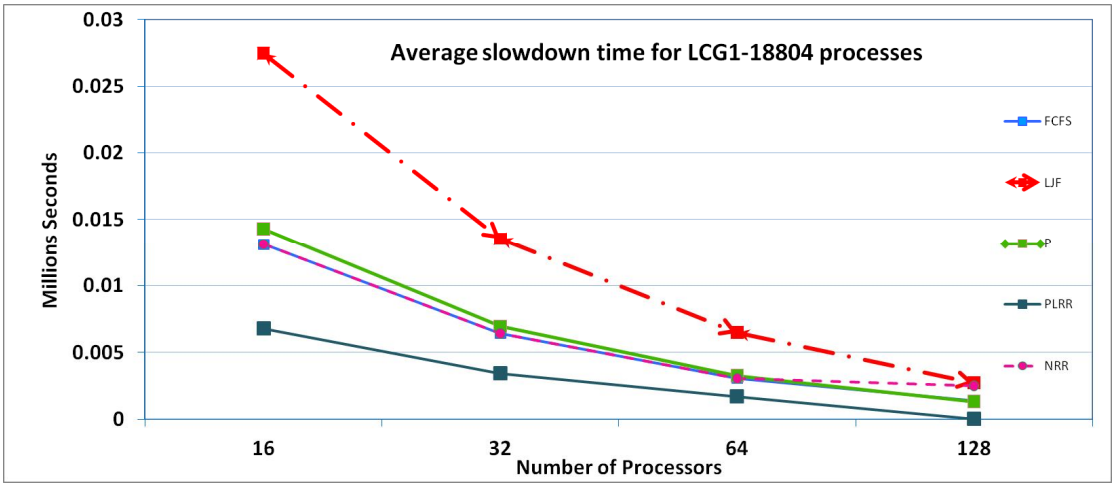

Figure 6.12(a): Average slowdown times of five algorithms for 10% workload of LCG1

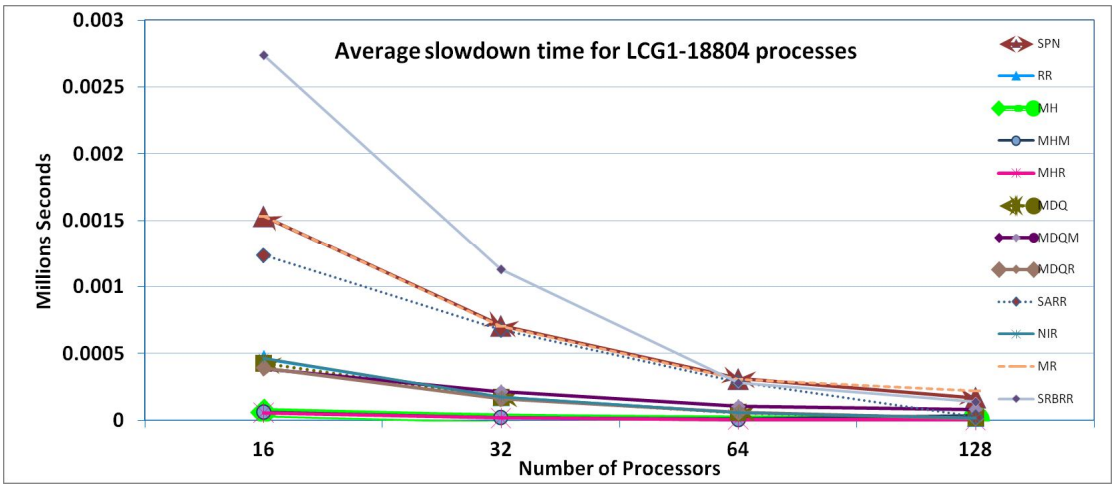

Figure 6.12(b): Average slowdown times of twelve algorithms for 10% workload of LCG1

Figure 6.12 (a, b) shows the average slowdown times computed for each scheduling algorithm using '10%' workload of LCG1. Figure 6.12 (a, b) shows that MH and MHR have produced the shortest average slowdown times compared to other scheduling algorithms. Figure 6.12 (a, b) also presents that MHM, RR, NIR, MDQ, MDQR and SARR have shown better performance w. r. t. the average slowdown times. Figure 6.12 (a, b) also presents that MDQM, SPN, MR and SRBRR have shown average performance w. r. t. the average slowdown times. It has also shown that PLRR, P, NRR, FCFS and LJF have shown the worst performance while resulting in longer average slowdown times. LJF has shown the longest average slowdown times. As a result, MH and MHR have shown the best average slowdown times compared to other scheduling algorithms and presented improvement w. r. t. average slowdown times under the increasing number of CPUs successively from '16' to '128'.

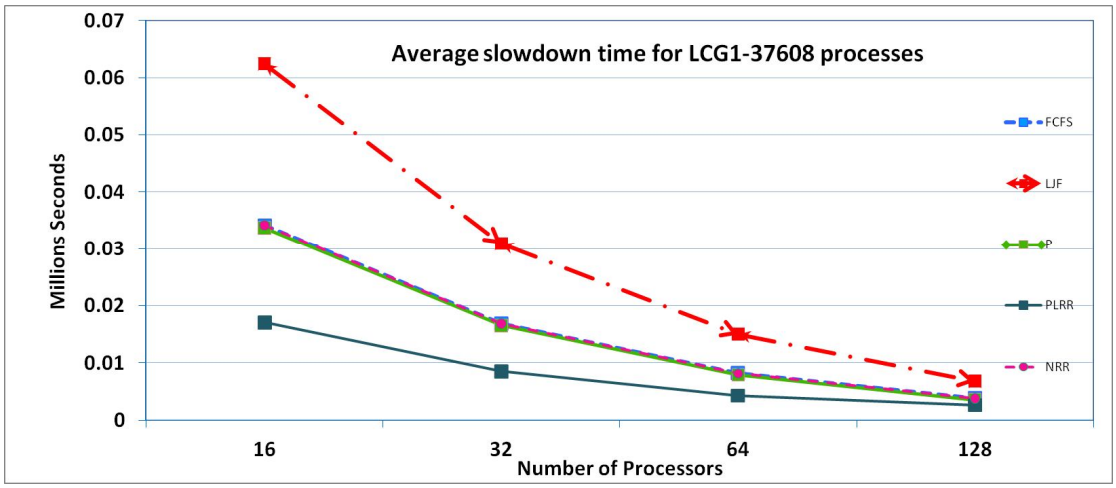

Figure 6.12(c): Average slowdown times of five algorithms for 20% workload of LCG1

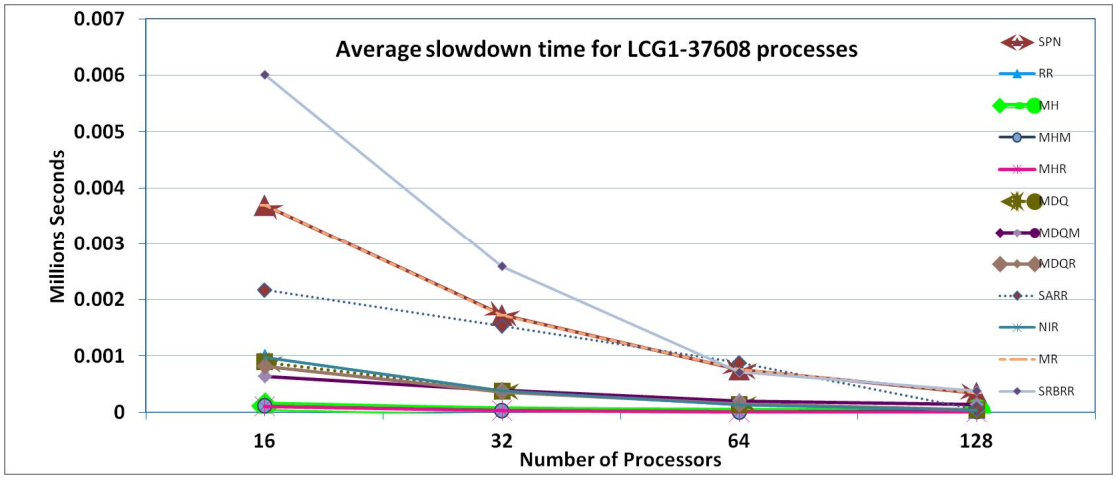

Figure 6.12(d): Average slowdown times of twelve algorithms for 20% workload of LCG1

Figure 6.12(c, d) shows the average slowdown times computed for each scheduling algorithm using '20%' workload of LCG1. Figure 6.12(c, d) depicts that each algorithm has shown the same performance pattern, as seen and analyzed in Figure 6.12(a, b) for '10%' workload of LCG1. Figure 6.12(a, b, c, d) shows that all scheduling algorithms have shown the improvement in performance under the increasing number of CPUs successively from '16' to '128'. Moreover, these algorithms also have shown steady performance measure of average slowdown times under increased workload from '10%' to '20%' of LCG1. As a result, all scheduling algorithms support scalability under dynamic scheduling environment. Finally, MH and MHR have shown the best average slowdown times compared to other scheduling algorithms for both workload traces of LCG1.

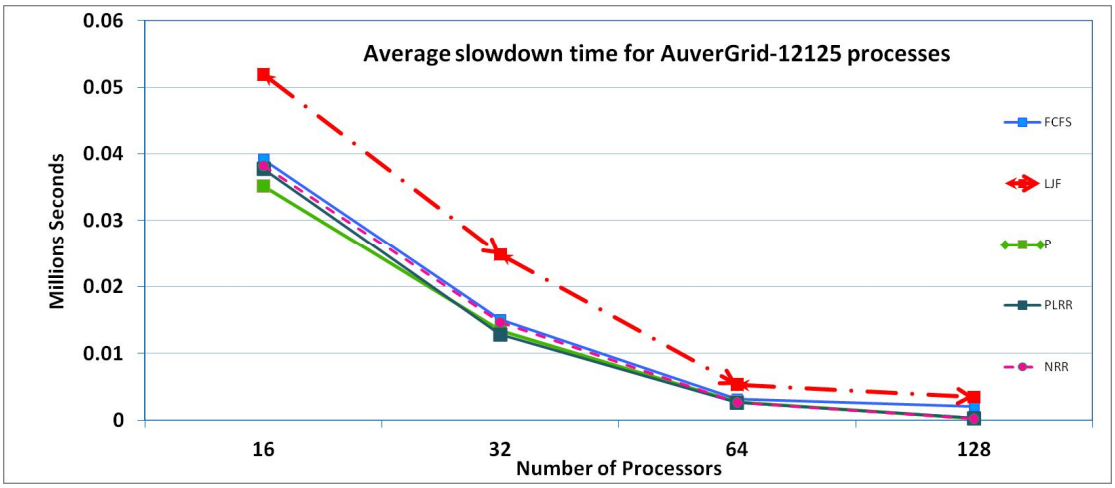

Figure 6.13(a): Average slowdown times of five algorithms for 3% workload of AuverGrid

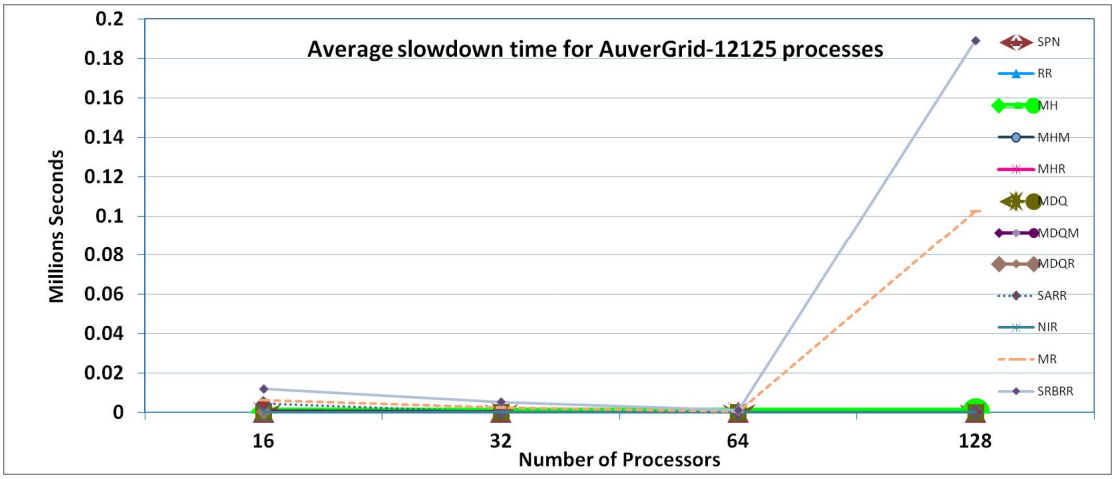

Figure 6.13(b): Average slowdown times of twelve algorithms for 3% workload of AuverGrid

Figure 6.13(a, b) shows the average slowdown times computed for each scheduling algorithm using '3%' workload of AuverGrid. Figure 6.13(a, b) shows that MH and MHR have produced the shortest average slowdown times compared to other scheduling algorithms. It also presents that MHM, RR, NIR, MDQ, MDQR and SARR have shown better performance w. r. t. the average slowdown times. Figure 6.13(a, b) also presents that MDQM and SPN have shown average results w. r. t. the average slowdown times. The MR and SRBRR have shown improvement in performance while increasing the number of CPUs from '16' to '64', and presented the poor performance on '128' CPUs. Figure 6.13(a, b) also shows that PLRR, P, NRR, FCFS and LJF have shown the worst performance while resulting longer slowdown times.

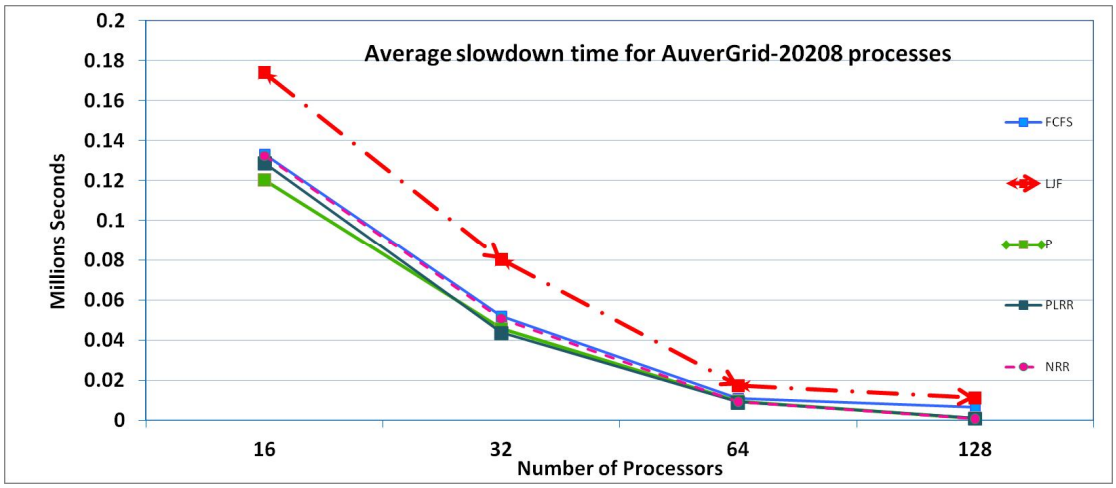

Figure 6.13(c): Average slowdown times of five algorithms for 5% workload of AuverGrid

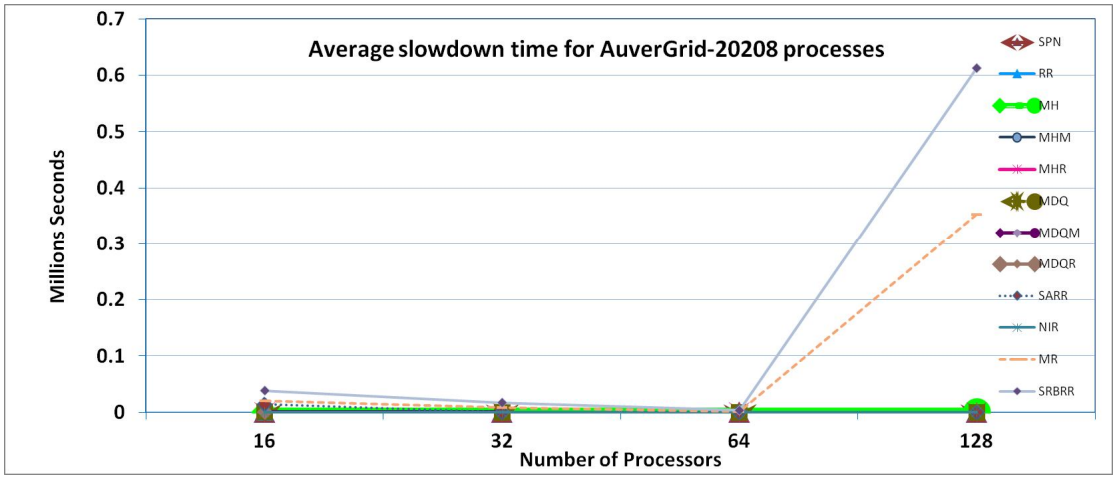

Figure 6.13(d): Average slowdown times of twelve algorithms for 5% workload of AuverGrid

Figure 6.13(c, d) shows the average slowdown times obtained for each scheduling algorithm using '5%' workload of AuverGrid. This figure also shows the same performance pattern w. r. t. average slowdown times as shown in Figure 6.13(a, b) for '3%' workload of AuverGrid. All scheduling algorithms have shown relative performance under the increasing workload of AuverGrid. Moreover, all scheduling algorithms, with the exception of SRBRR and MR, have shown that relatively better performance w. r. t. the average slowdown times, by increasing the number of CPUs successively from '16' to '128'. Figure 6.11(a, b, c, d)- Figure 6.13(a, b, c, d) shows that all scheduling algorithms, with the exception of SRBRR and MR, have shown the improvement in performance w. r. t. average slowdown times by increasing number of CPUs and also depicted maintained average slowdown times under different types of workload. As a result, MH and MHR have shown the best average slowdown times.

6.6.4.5 Total Completion Time Analysis

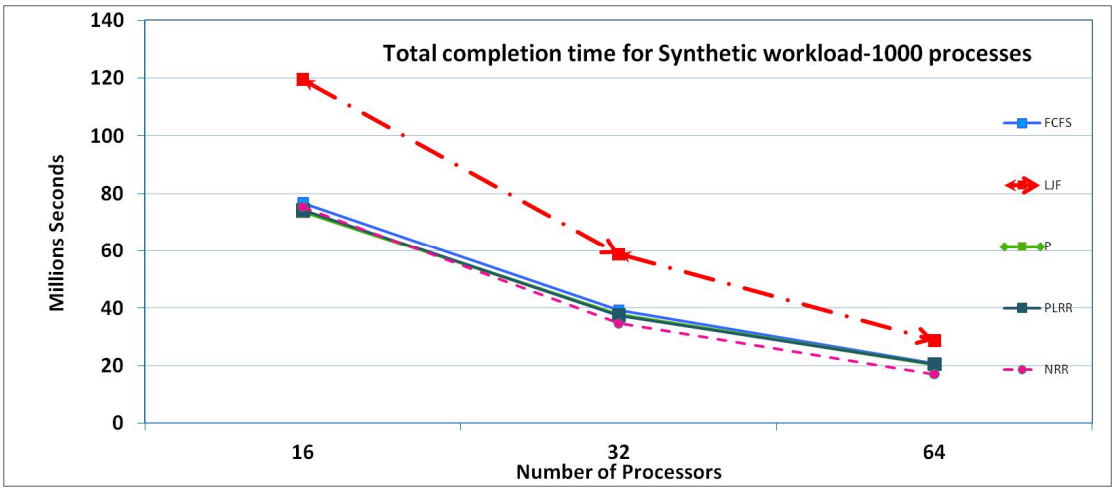

Figure 6.14(a): Total completion times of five algorithms for synthetic workload of 1000 processes

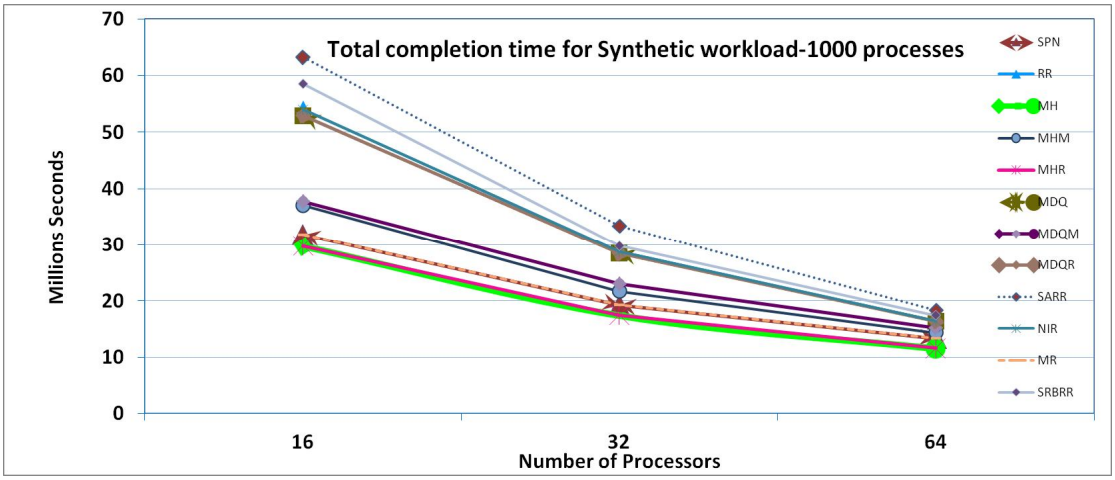

Figure 6.14(b): Total completion times of twelve algorithms for synthetic workload of 1000 processes

Figure 6.14(a, b) depicts that MH and MHR have produced the shortest total completion times compared to the other scheduling algorithms. Figure 6.14(a, b) also presents that SPN, MR, MHM and MDQM have shown better total completions times but higher than those for MH and MHR. Figure 6.14(a, b) also shows that MDQR, MDQ, RR, NIR, SARR and SRBRR have shown longer total completion times. Figure 6.14(a, b) also shows that PLRR, P, NRR, FCFS and LJF have shown the longest completion times and hence the performance is worst. LJF has resulted in the longest completion times. Moreover, all scheduling algorithms have shown improvement in total completion times by increasing the number of CPUs successively from '16' to '64'.

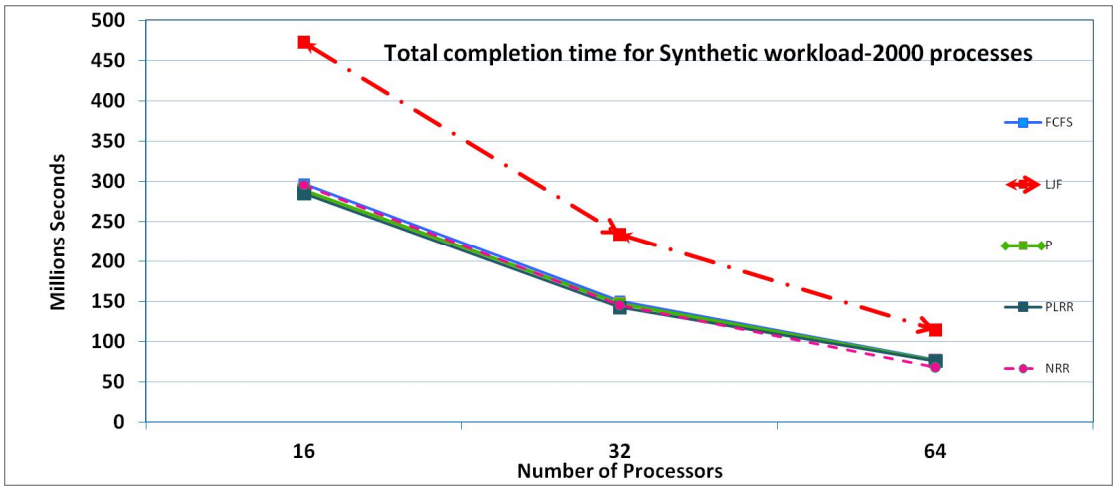

Figure 6.14(c): Total completion times of five algorithms for synthetic workload of 2000 processes

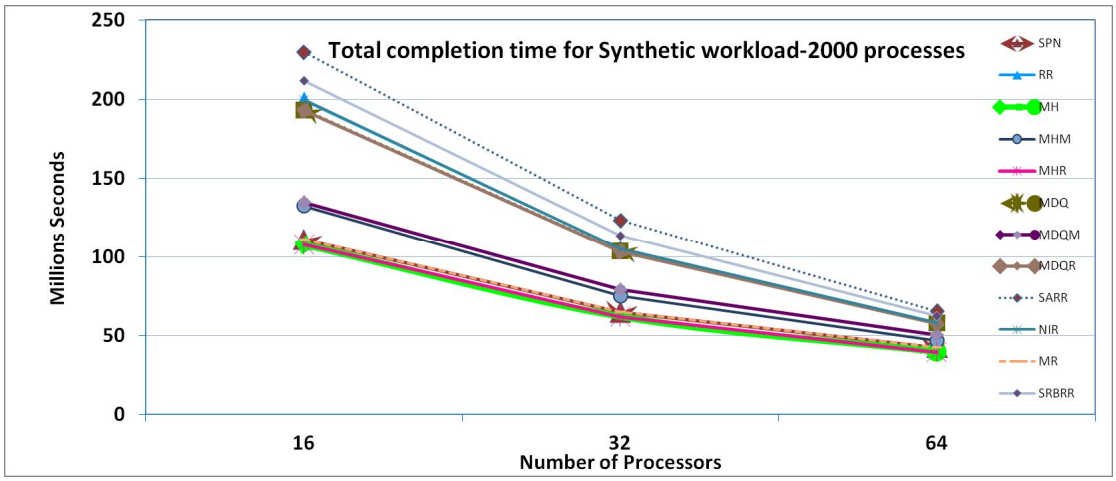

Figure 6.14(d): Total completion times of twelve algorithms for synthetic workload of 2000 processes

Figure 6.14 (c, d) shows the total completion times obtained for each scheduling algorithm using '2000' process of synthetic type. Figure 6.14(c, d) shows the same performance pattern of each scheduling algorithm as observed and analyzed in Figure 6.14(a, b). All scheduling algorithms have shown relative performance under the increasing synthetic workload from '1000' to '2000' processes.

Figure 6.14(a, b, c, d) shows that all scheduling algorithms support scalability under the increased synthetic workload from '1000' to '2000' processes, and varying the number of CPUs successively from '16' to '64' under dynamic scheduling environment. Finally, MH and MHR have shown the best total completion times compared to other scheduling algorithms using synthetic workload traces.

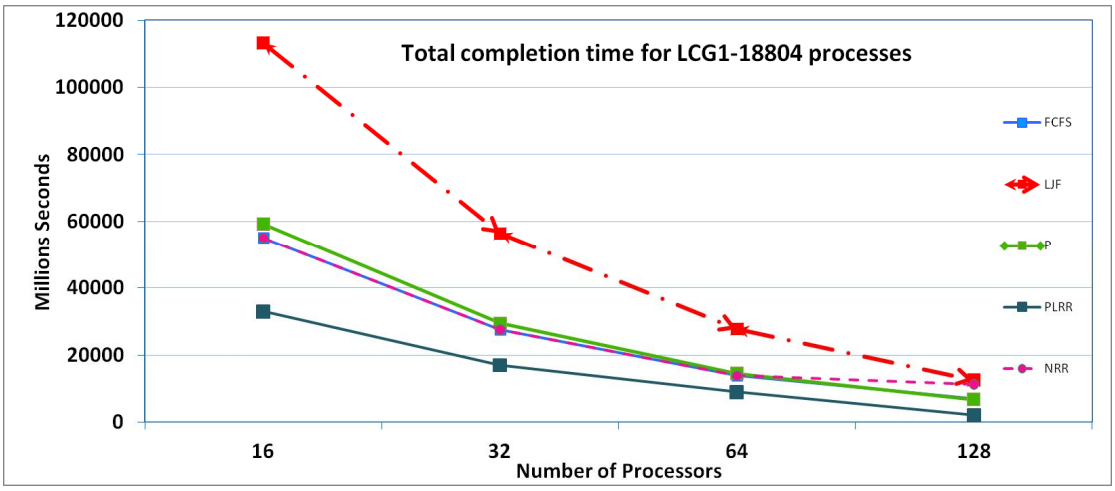

Figure 6.15(a): Total completion times of twelve algorithms for 10% workload of LCG1

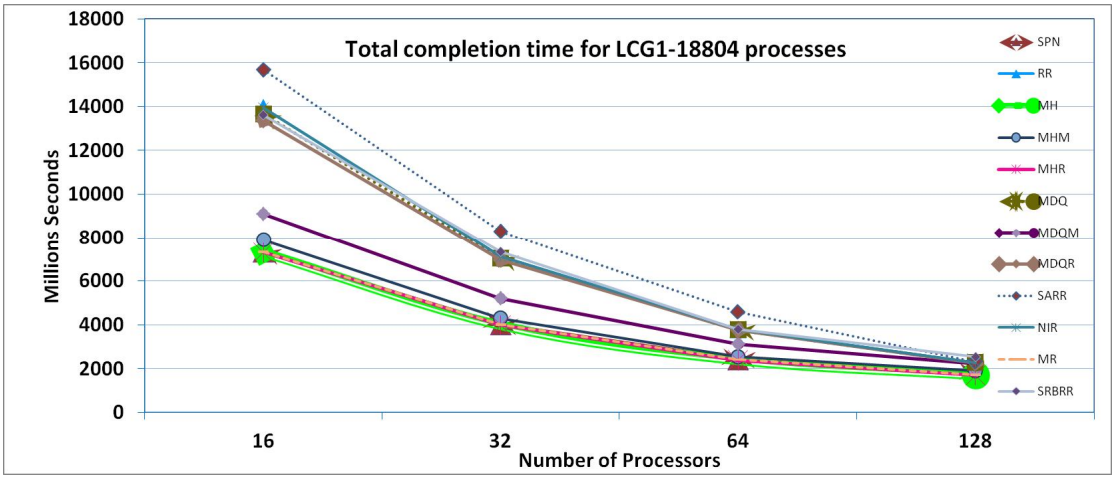

Figure 6.15(b): Total completion times of twelve algorithms for 10% workload of LCG1

Figure 6.15(a, b) shows the total completion times computed for each scheduling algorithm using '10%' workload of LCG1. Figure 6.15(a, b) shows that MH and MHR have produced the shortest total completion times compared to the other scheduling algorithms. Figure 6.15(a, b) also presents that SPN, MR, MHM and MDQM have shown slightly higher total completion times than those for MH and MHR. Figure 6.15(a, b) also presents that MDQR, MDQ, RR, NIR, SARR and SRBRR have shown longer total completion times. Figure 6.15(a, b) also depicts that PLRR, P, NRR, FCFS and LJF have shown the worst performance, resulting in longer completion times. Moreover, all scheduling algorithms have shown improvement in total completion times by increasing the number of CPUs for synthetic workload trace. As a result, MH and MHR have shown best total completion times for '10%' workload of LCG1.

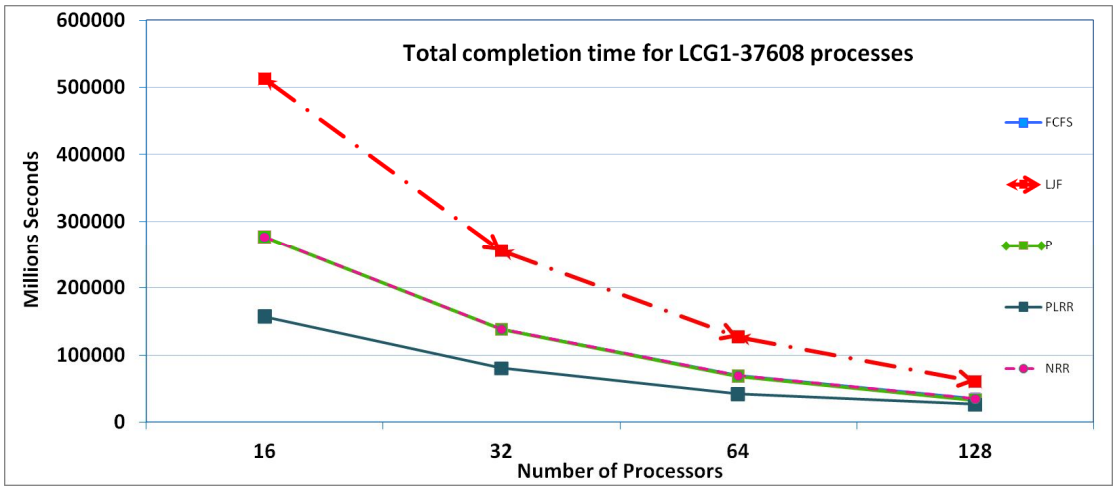

Figure 6.15(c): Total completion times of twelve algorithms for 20% workload of LCG1

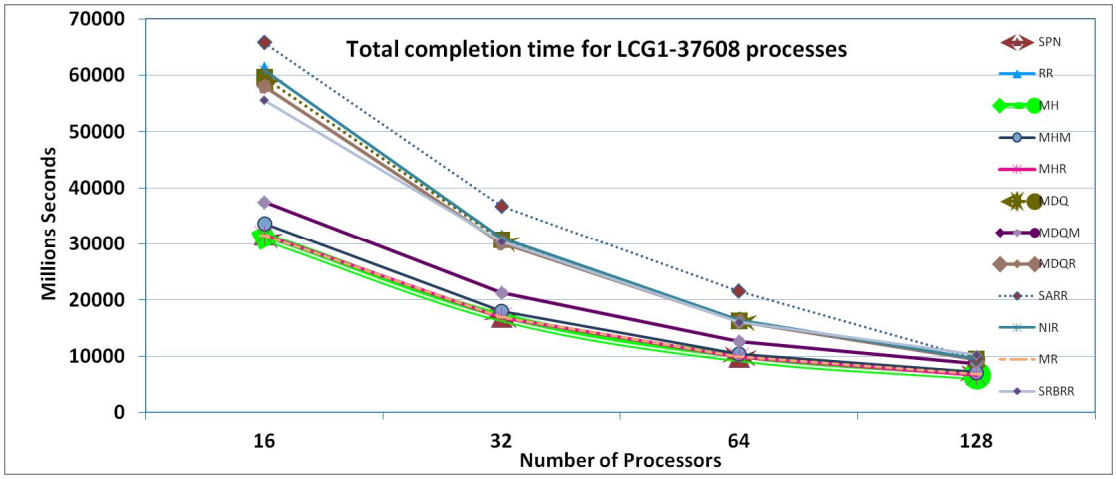

Figure 6.15(d): Total completion times of five algorithms for 20% workload of LCG1

Figure 6.15(c, d) shows the total completion times computed for each scheduling algorithm using '20%' workload of LCG1. Results has shown the same performance pattern of total completion times for each scheduling algorithm as seen and analyzed in Figure 6.15(a, b) for '10%' workload of LCG1. Figure 6.15(a, b, c, d) shows that all scheduling algorithms have shown reduced total completion times with increasing number of CPUs successively from '16' to '128'.

Moreover, all scheduling algorithms have shown relative performance under the increasing workload of LCG1, i.e., from '10%' to '20%' of it. As a result, all scheduling algorithms have shown true scalability under dynamic scheduling environment. Finally, MH and MHR have shown the best results w. r. t. the total completion times compared to the other scheduling algorithms, under increased workload traces of LCG1 and by varying the number of CPUs progressively.

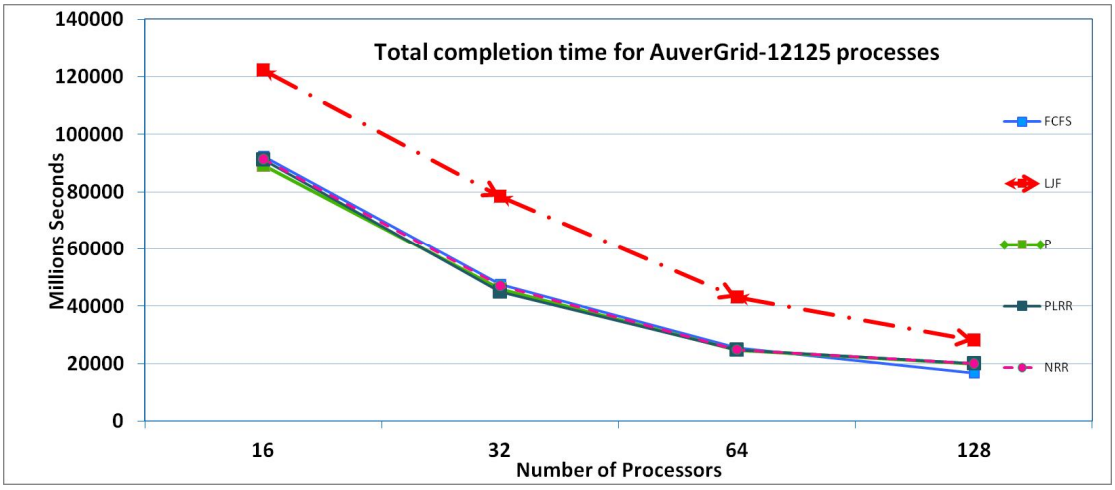

Figure 6.16(a): Total completion times of five algorithms for 3% workload of AuverGrid

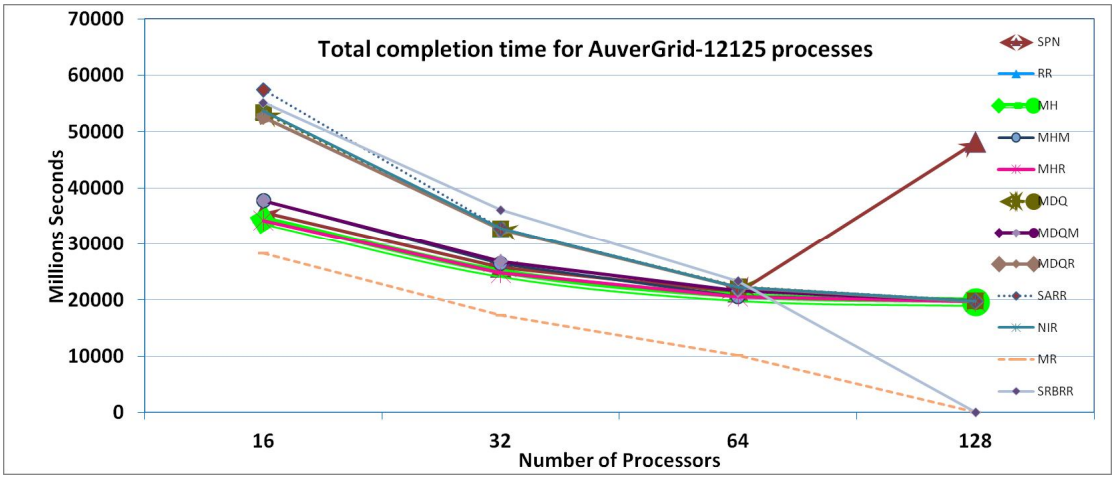

Figure 6.16(b): Total completion times of twelve algorithms for 3% workload of AuverGrid

Figure 6.16(a, b) shows that MR has shown the shortest total completion times compared to other scheduling algorithms. This figure also depicts that MH and MHR have shown the longer completion times than the values for MR but shorter completion times than the values for other scheduling algorithms. Figure 6.16(a, b) also presents that MHM and MDQM have shown longer total completion times than MR, MH and MHR. At '128' CPUs, SPN has shown the worst total completion times because load balancing algorithm has not been applied for workload distribution among grid nodes. Some of the processors have become heavily loaded for execution, resulted in the longer total completion times. This figure also presents that MDQR, MDQ, RR, NIR, SARR and SRBRR have shown longer total completion times than MH, MHR, MR, MHM and MDQM. This figure also depicts that PLRR, P, NRR, FCFS and LJF have shown the worst performance w. r. t. total completion times.

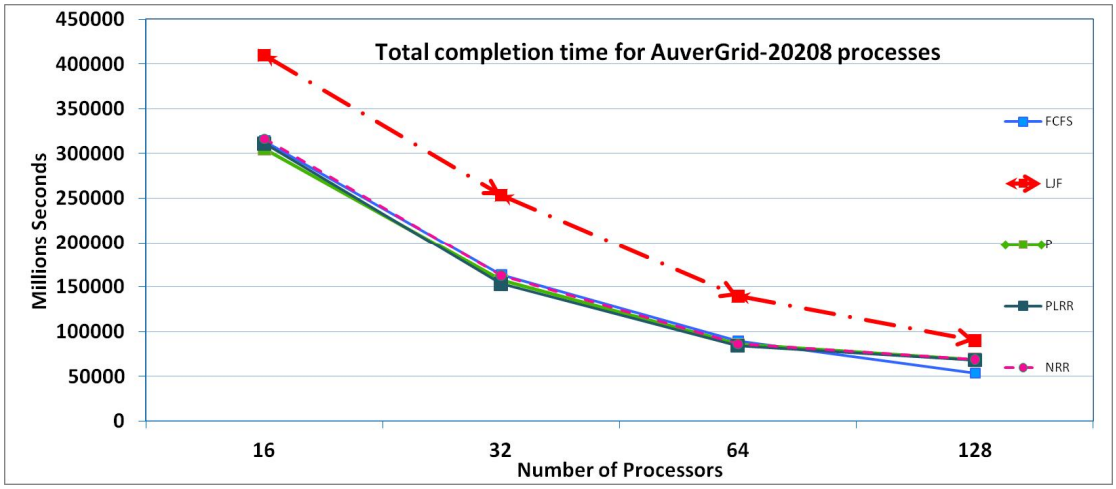

Figure 6.16(c): Total completion times of five algorithms for 5% workload of AuverGrid

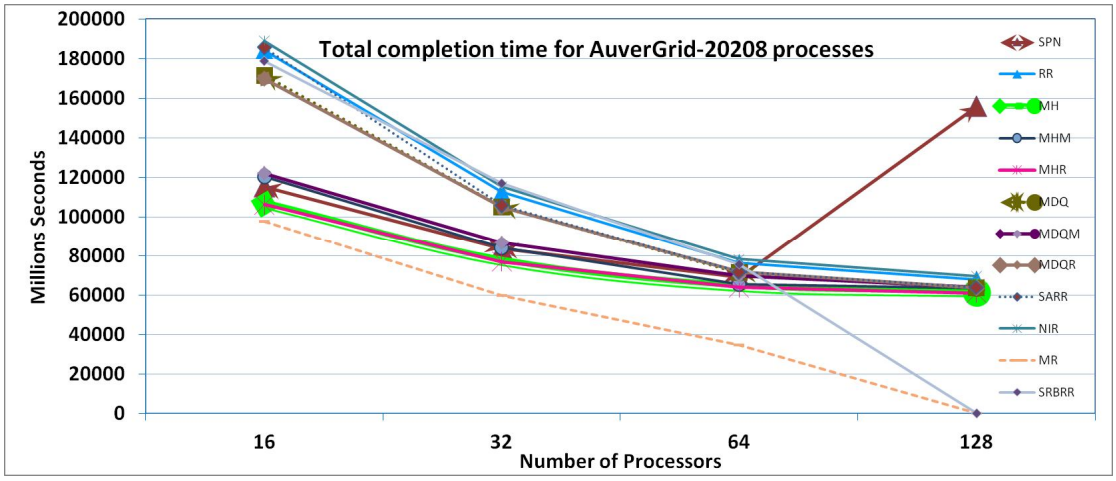

Figure 6.16(d): Total completion times of twelve algorithms for 5% workload of AuverGrid

Figure 6.16(c, d) has shown the same performance pattern w. r. t. total completion times as observed in Figure 6.16(a, b). All scheduling algorithms have shown relative total completion times for 3-5% of AuverGrid workload using '16' to '128' CPUs. As a result, MR has shown the best total completion times. The values of total completion times for MH and MHR are longer than the values for MR but shorter than the values for other scheduling algorithms. Figure 6.14(a, b, c, d) and Figure 6.15(a, b, c, d) shows that MH and MHR have shown best total completion times for synthetic and LCG1 workload traces whilst Figure 6.16(a, b, c, d) represents that MR has shown the best total completion times for AuverGrid traces. SRBRR also has shown the best total completion times using '128' CPUs for AuverGrid workload traces. Moreover, all scheduling algorithms, with the exception of SPN, show that relative total completion time independent of workload conditions and support scalability.

6.6.5.6 Maximum Job Stretch Time Analysis

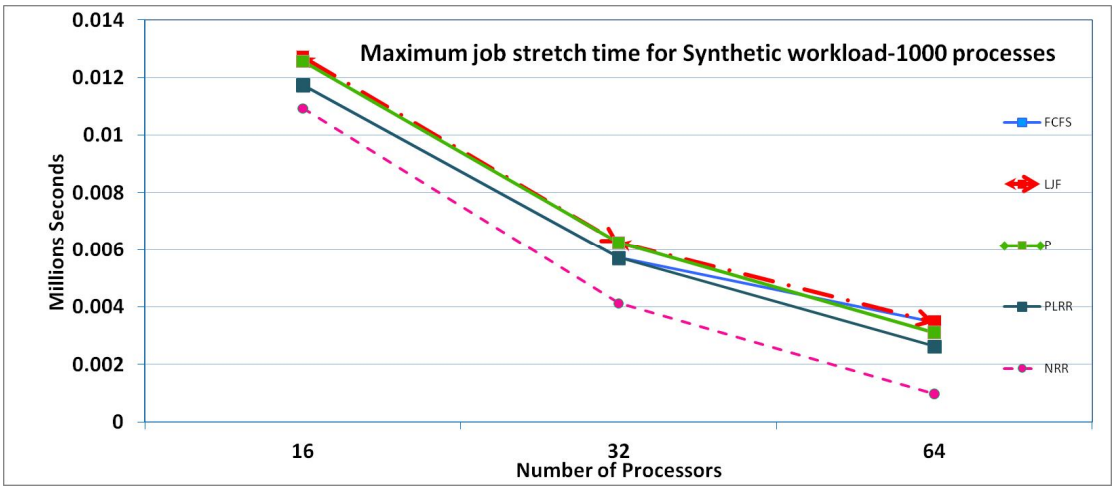

Figure 6.17(a): Maximum Job Stretch times of five algorithms for synthetic workload of 1000 processes

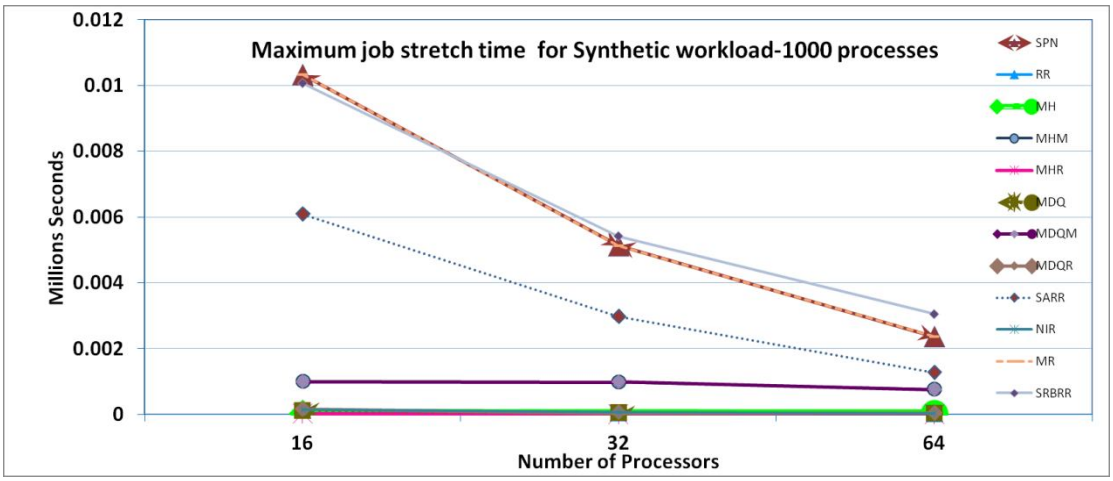

Figure 6.17(b): Maximum Job Stretch times of twelve algorithms for synthetic workload of 1000 processes

Figure 6.17(a, b) show the maximum job stretch times computed using synthetic workload trace of '1000' processes. Figure 6.17(a, b) shows that MH and MHR have produced the best maximum job stretch times compared to the other scheduling algorithms. Figure 6.17(a, b) also shows that RR, NIR, MDQ, MDQR, SARR, MHM and MDQM have shown better maximum job stretch times but longer than those for MH and MHR. Figure 6.17(a, b) also depicts that SPN, MR and SRBRR have shown average results w. r. t. the maximum job stretch times. Figure 6.17(a, b) also shows that P, PLRR, NRR, FCFS and LJF have shown the worst maximum job stretch times. Moreover, all scheduling algorithms have shown improvement in maximum job stretch times under the increasing number of CPUs successively from '16' to '64' for synthetic workload of '1000' processes.

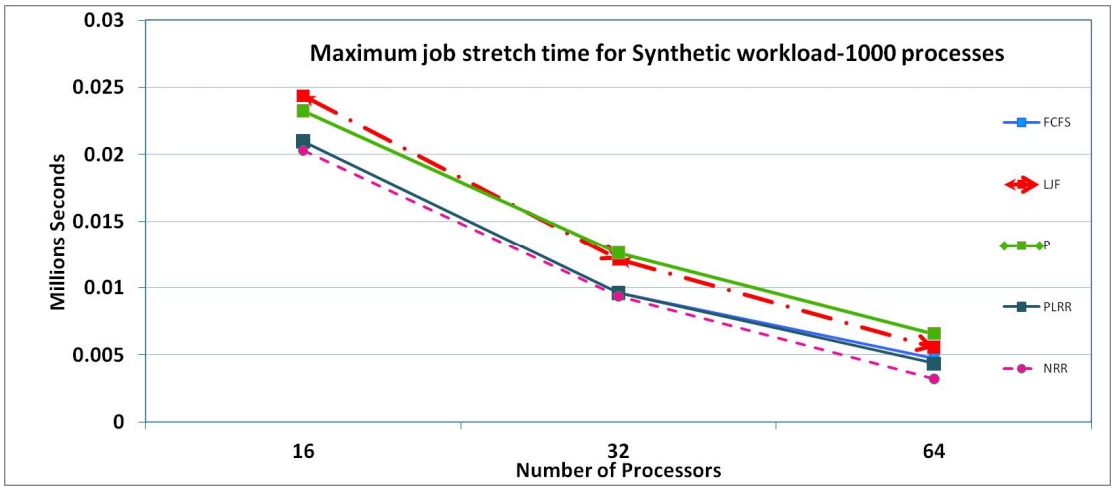

Figure 6.17(c): Maximum Job Stretch times of five algorithms for synthetic workload of 2000 processes

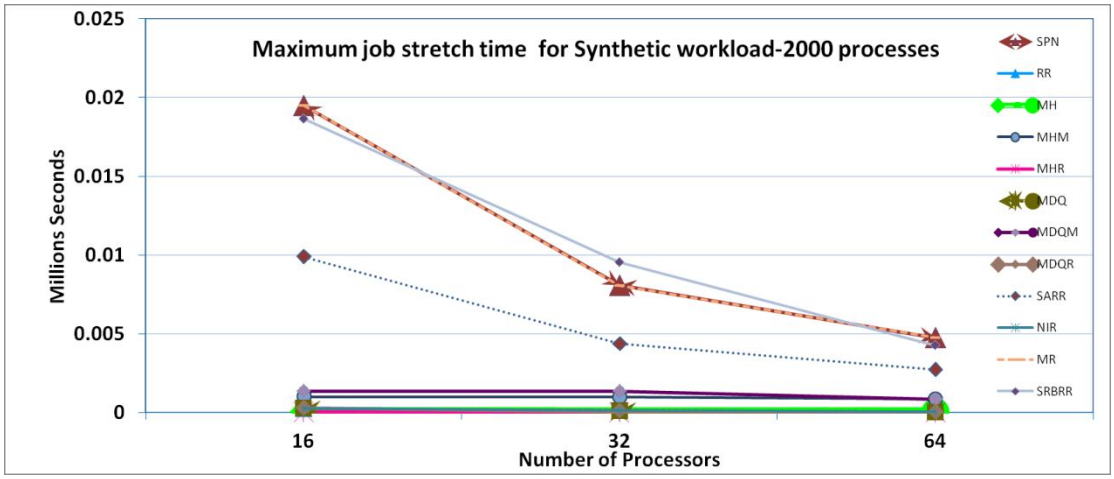

Figure 6.17(d): Maximum Job Stretch times of twelve algorithms for synthetic workload of 2000 processes

Figure 6.17(c, d) shows the maximum job stretch times computed for each scheduling algorithm using synthetic workload of '2000' processes. Figure 6.17(c, d) shows the same result pattern for the maximum job stretch times as observed in Figure 6.17(a, b) for the synthetic workload of '1000' processes. It means all scheduling algorithms have shown relative performance w. r. t. maximum job stretch times for increased synthetic workload traces. In addition, all scheduling algorithms have shown improvement in maximum job stretch times by increasing the number of CPUs successively from '16' to '64'. As a result, all scheduling algorithms have shown scalability under dynamic scheduling environment; and MH and MHR have shown the best maximum job stretch times for synthetic workload traces.

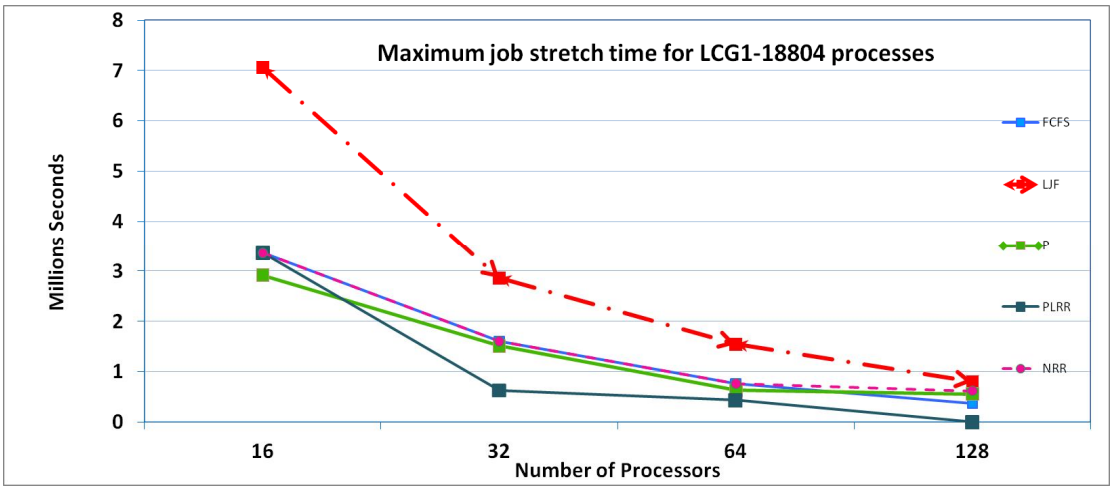

Figure 6.18(a): Maximum Job Stretch times of five algorithms for 10% workload of LCG1

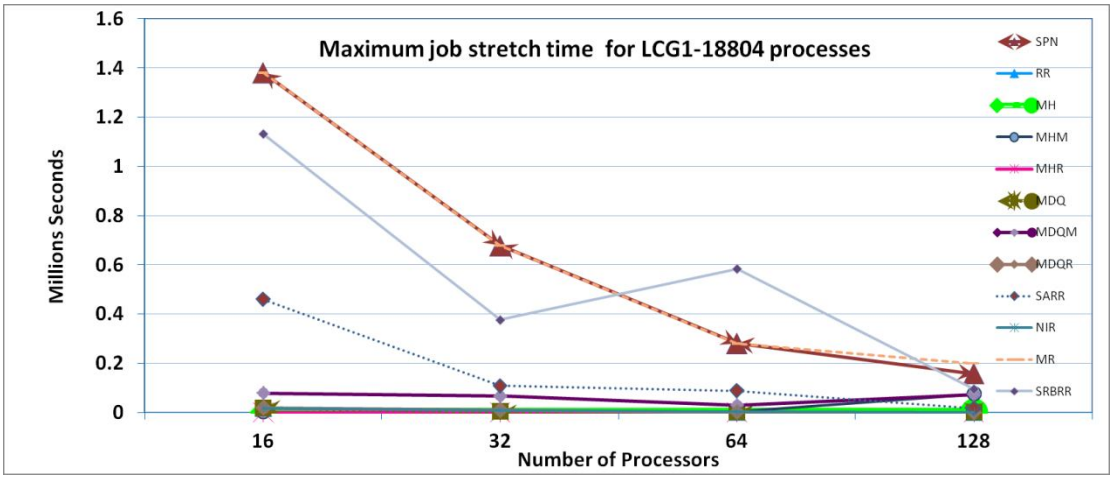

Figure 6.18(b): Maximum Job Stretch times of twelve algorithms for 10% workload of LCG1

The maximum job stretch times for each scheduling algorithm using '10%' workload of LCCG1 are shown in Figure 6.18(a, b). It can be depicted that MH and MHR have shown the shorter maximum job stretch times compared to the other scheduling algorithms. In addition, RR, NIR, MDQ, MDQR, SARR, MHM and MDQM have shown the average measures of maximum job stretch times. SPN and MR have shown the longer values for total completion times. However, Figure 6.18(a, b) shows fluctuation in experimental values presenting the performance of SRBRR. The SRBRR has shown the worst maximum job stretch times on 64 CPUs. Figure 6.18(a, b) also shows that P, PLRR, NRR, FCFS and LJF have produced the longest maximum job stretch times. As a result, all scheduling algorithms except SRBRR have shown the improvement in maximum job stretch times for '10%' workload of LCG1. In addition, MH and MHR have shown the best maximum job stretch times.

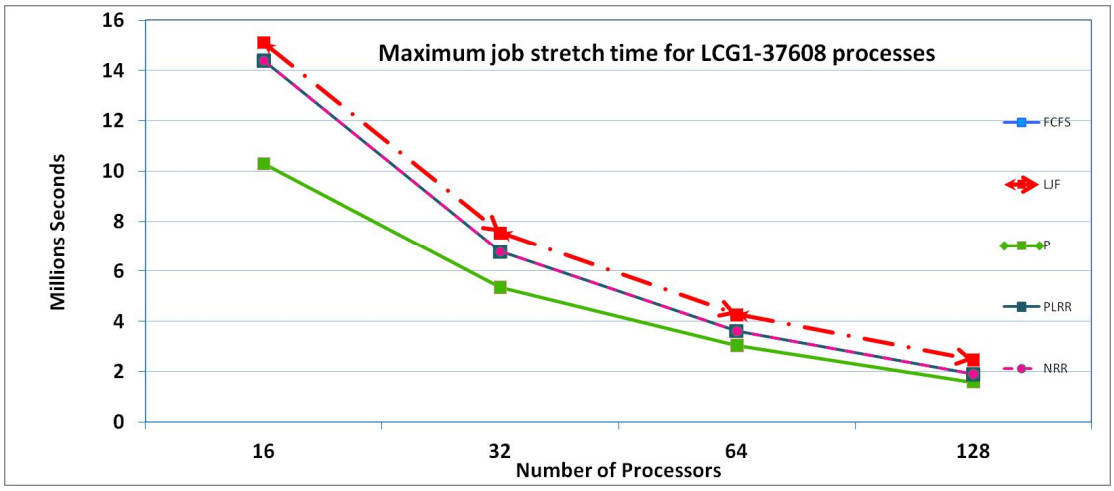

Figure 6.18(c): Maximum Job Stretch times of five algorithms for 20% workload of LCG1

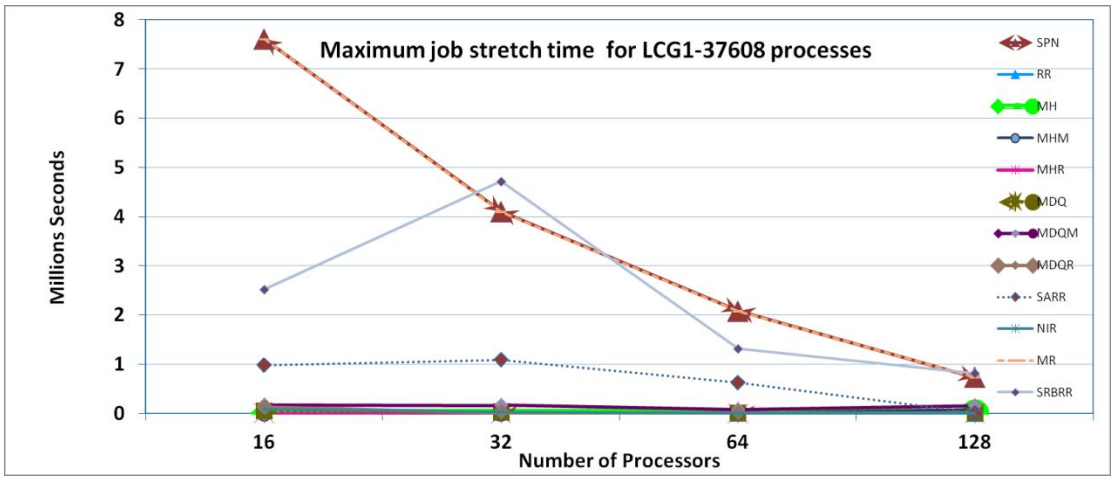

Figure 6.18(d): Maximum Job Stretch times of twelve algorithms for 20% workload of LCG1

Figure 6.18(c, d) shows the maximum job stretch times computed for each scheduling algorithm using '20%' workload of LCG1. Figure 6.18(c, d) depicts all scheduling algorithms, with the exception of SRBRR and SARR, have shown the improvement in maximum job stretch times under the increasing number of CPUs successively from '16' to '128'. All scheduling algorithms, except SRBRR and SARR, have shown the same performance pattern presenting maximum job stretch times as observed in Figure 6.18(a, b) for '10%' workload of LCG1. All scheduling algorithms, except SRBRR and SARR, have shown steady maximum job stretch times under the increasing workload of LCG1 and represented improvement in values of job stretch times by increasing the number of CPUs. As a result, MH and MHR have shown the best maximum job stretch times for traces of LCG1.

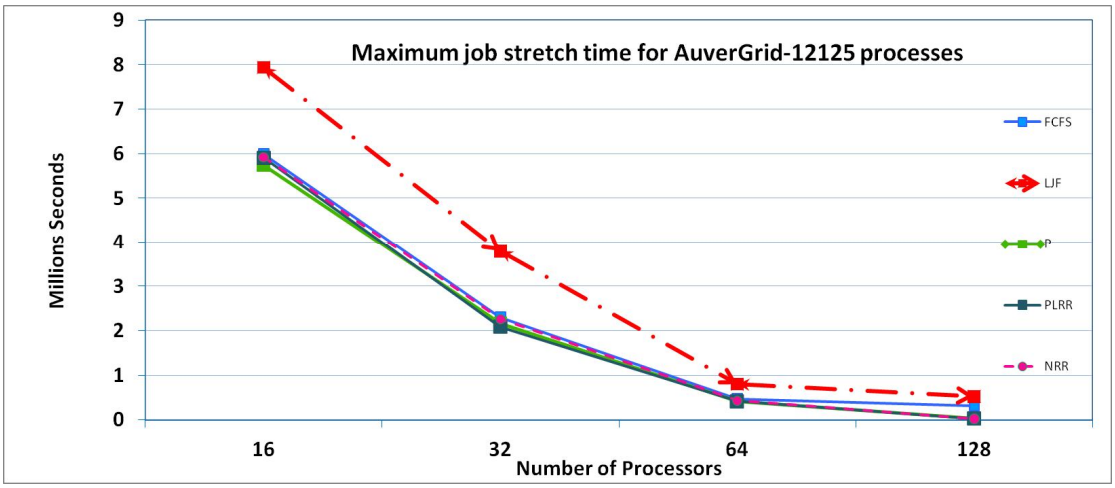

Figure 6.19(a): Maximum Job Stretch times of five algorithms for 3% workload of AuverGrid

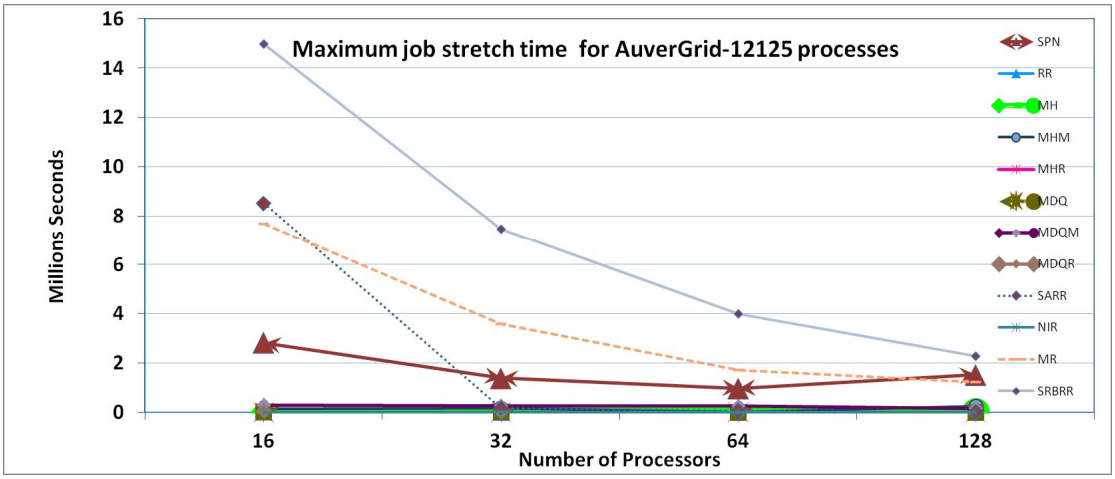

Figure 6.19(b): Maximum Job Stretch times of twelve algorithms for 3% workload of AuverGrid

Figure 6.19(a, b) presents that MH and MHR have produced the shortest maximum job stretch times compared to the other scheduling algorithms. Moreover, Figure 6.19(a, b) also shows that RR, NIR, MDQ, MDQR, SARR, MHM and MDQM have shown better maximum job stretch times but longer than the values computed by MH and MHR. Figure 6.19(a, b) also depicts that SPN, MR and SRBRR have shown average measures for the maximum job stretch times. Figure 6.19(a, b) also shows the fluctuation in line for presenting the maximum job stretch times for SPN. The SPN has shown the worst performance on '128' CPUs. Figure 6.19(a, b) also present that P, PLRR, NRR, FCFS and LJF have produced the longest maximum job stretch times. As a result, MH and MHR have shown the best maximum job stretch times by increasing the number of CPUs successively from '16' to '128', using '3%' workload of AuverGrid.

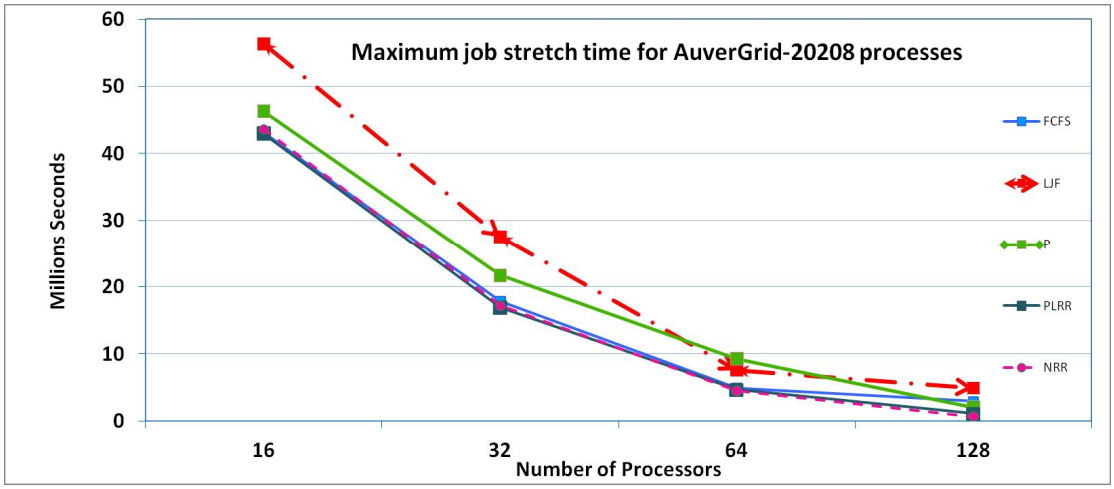

Figure 6.19(c): Maximum Job Stretch times of five algorithms for 5% workload of AuverGrid

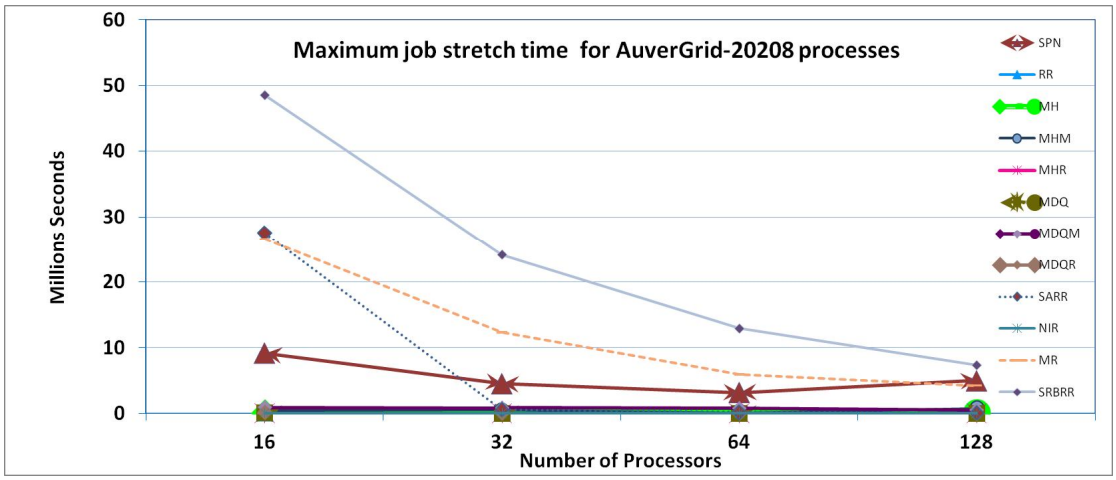

Figure 6.19(d): Maximum Job Stretch times of twelve algorithms for 5% workload of AuverGrid

Figure 6.19(c, d) shows that each scheduling algorithm has shown the same pattern for maximum job stretch times as observed in Figure 6.19(a, b) for '3%' workload of AuverGrid. It means all scheduling algorithms, except SPN, have shown the relative measures of maximum job stretch times under the increasing workload of AuverGrid. Figure 6.17(a, b, c, d)- Figure 6.19(a, b, c, d) have concluded that MH and MHR have shown best maximum job stretch times under different types of workload and varied number of CPUs. It is also clear from this figure that P, PLRR, NRR, FCFS and LJF have shown the worst maximum job stretch times. Finally all scheduling algorithms, except SPN, have shown the support for scalability for maximum job stretch times under the various workload conditions and by varying the number of CPUs successively from '16' to '128'.

6.6.4.7 Performance Analysis of Scheduling Algorithms by Changing Time Quantum

The RR, MH and MDQ scheduling algorithms work on a fixed time quantum value. It has been shown that very small value of fixed time quantum will result in improved average response time but it may produce many context switches. At the same time, very high value of the fixed time quantum will result in the less efficient performance. Proposed dynamic scheduling algorithms namely MHM, MHR, MDQM and MDQR use a dynamic time quantum strategy instead of a static one and they maintain system performance in dynamic scheduling environment. The value of the time quantum is computed at runtime considering the runtime demands of user jobs as well as the total number of present jobs in the system. Performance parameters have been computed using each scheduling algorithm for '10%' workload of LCG1 using 64 processors and varying the time quantum from '50' to '5000' seconds as shown in Figure 6.20.

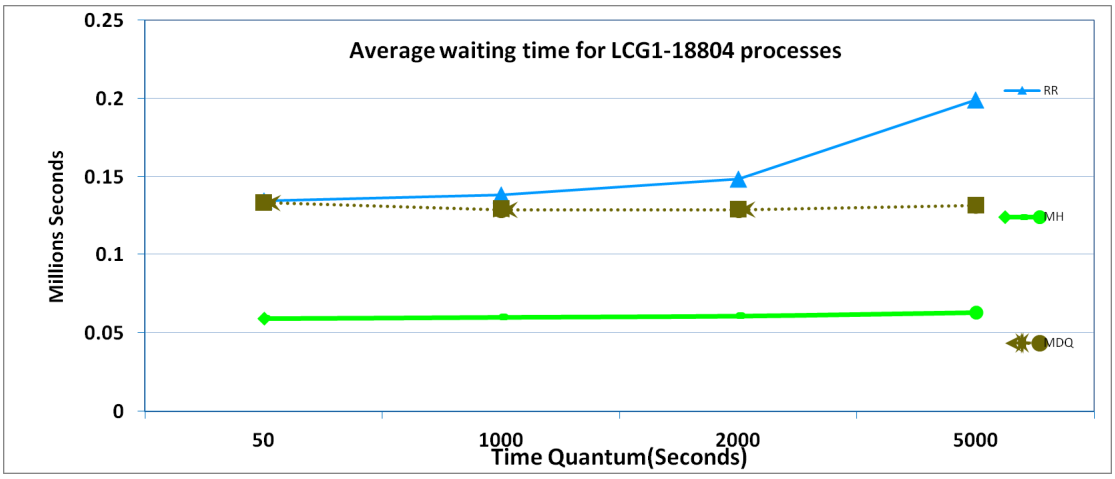

Figure 6.20: Average waiting times of scheduling algorithms by changing the Time Quantum

Figure 6.20 shows the average waiting times for RR, MH and MDQ scheduling algorithms under the fixed time quantum in the range of '50' to '500' seconds. Each scheduling algorithm has resulted longer average waiting times by increasing the values of time quantum. MH has shown the best average waiting times, as compared to RR and MDQ, for workload trace of LCG1. RR has shown the worst performance w. r. t. waiting times under increased values of time quantum. As a result, MH, MDQ and RR have shown longer average waiting times under the increasing values of time quantum.

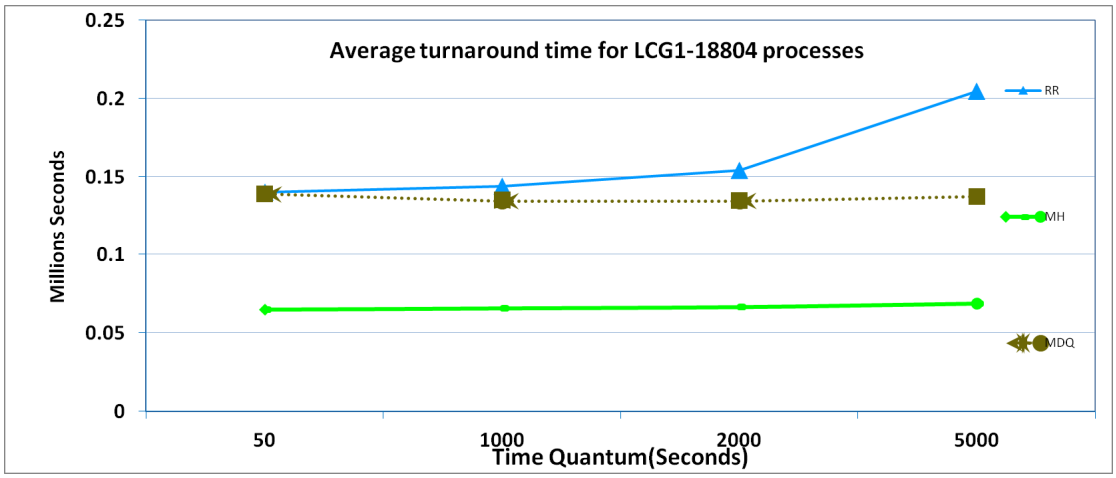

Figure 6.21: Average turnaround times of scheduling algorithms by changing the Time Quantum

Figure 6.21 shows the average turnaround times for RR, MH and MDQ using time quantum in the range of '50' to '500'. Each of these scheduling algorithms has shown longer average turnaround times under the increasing values of time quantum. RR has shown the worst performance at time quantum of '5000' seconds. MH and MDQ have also shown poor performance while representing higher values with increase of time quantum value.

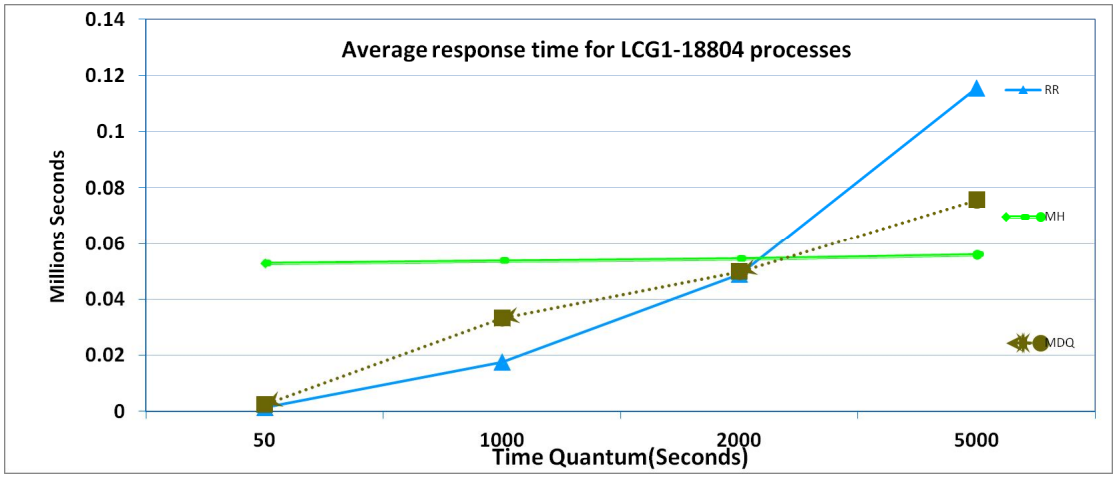

Figure 6.22: Average response times of scheduling algorithms by changing the Time Quantum

Average response times for RR, MH and MDQ scheduling algorithms have been portrayed using different values of time quantum in Figure 6.22. Each scheduling algorithm has resulted in longer average response times with increase in the values of time quantum. RR and MDQ have shown the best performance at '50' seconds of time quantum and worst performance at '5000' seconds. MH also has shown longer

average response times (i.e., 52955.7157, 53836.9771, 54642.5696 and 56083.6974 seconds) under the increasing time quantum from ‘50’ to ‘5000’ seconds.

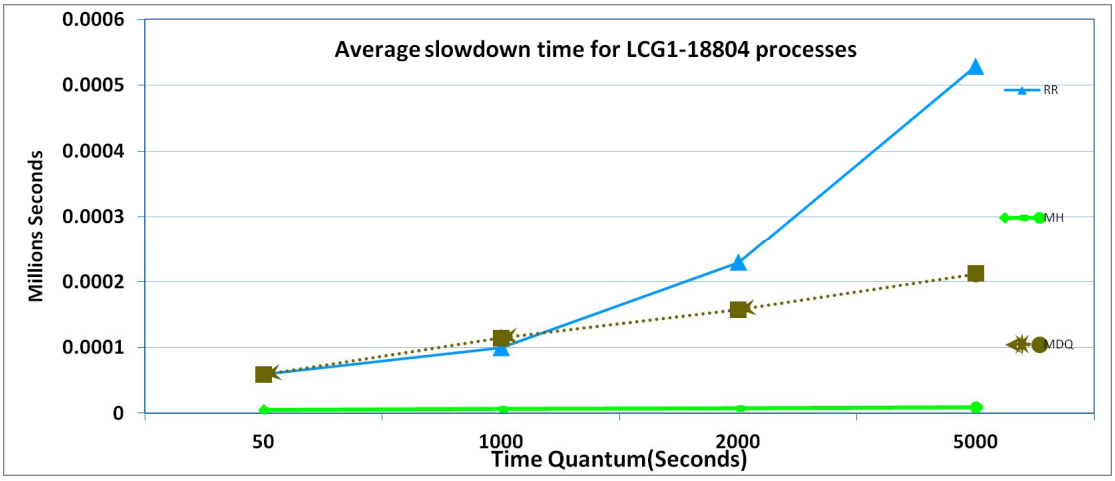

Figure 6.23: Average slowdown times of scheduling algorithms by changing the Time Quantum

Figure 6.23 shows the average slowdown times of RR, MH and MDQ scheduling algorithms using ‘10%’ workload of LCG1 under varied value of time quantum from ‘50’ to ‘5000’ seconds. Each algorithm has shown the longer average slowdown times under the increasing value of time quantum. RR has shown the worst performance w. r. t. slowdown times whilst MH represented the best at time quantum of ‘5000’ seconds

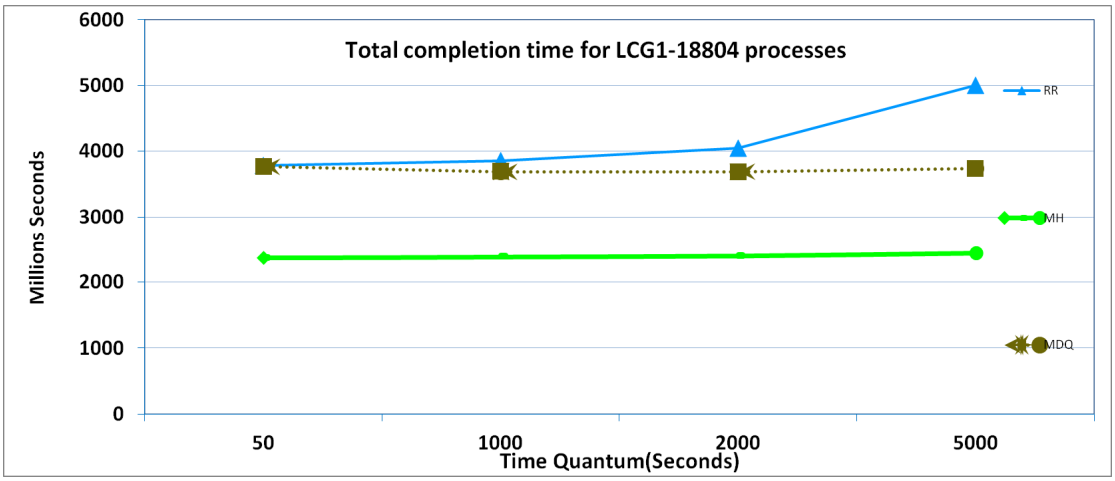

Figure 6.24: Total completion times of scheduling algorithms by changing the Time Quantum

Figure 6.24 shows the total completion times for RR, MH and MDQ scheduling algorithms under increasing value of time quantum from ‘50’ to ‘500’. Each algorithm has resulted longer total completion times with the increase of time

quantum. MH has shown the best total completion times, as compared to RR and MDQ, for workload trace of LCG1. RR has shown the worst total completion times under increased values of time quantum. As a result, MH, MDQ and RR have shown longer average total completion times by increasing the value of time quantum.

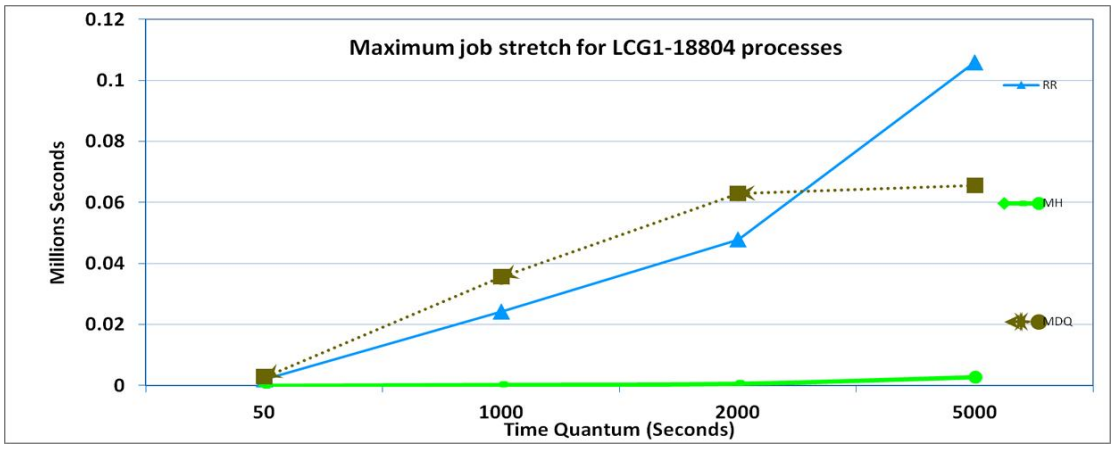

Figure 6.25: Job stretch times of scheduling algorithms by changing the Time Quantum

Figure 6.25 illustrates the job stretch times of RR, MH and MDQ scheduling algorithms using '10%' workload of LCG1 under the increased value of time quantum from '50' to '5000' seconds. Each algorithm has shown the longer job stretch times with increase of time quantum value. MH has shown the best job stretch times whilst RR has shown the worst at time quantum of '5000' seconds. Moreover, RR, MH and MDQ have shown longer job stretch times by varying the value of time quantum. Finally, RR, MH and MDQ have shown poor performance under the increasing values of time quantum for '10%' workload of LCG1. Out of these three, MH has shown the best whilst RR has shown the worst results for all performance parameters.

6.6.5.8 Summary of Performance Analysis

The detailed analysis of proposed and other scheduling algorithms have been performed under different workload conditions in dynamic scheduling environment. Section 6.6.5.1- section 6.6.5.7 reveals that the proposed scheduling algorithms namely MH, MHR and MDQ are acceptable for the scheduling of jobs on computational grid. These proposed ones have the capabilities to replace the existing approaches as they have shown the significant improvement for all performance factors. The experimental results based on the performance evaluation criteria have

shown that MH and MHR have shown the best average waiting time, average turnaround time, average slowdown time and job stretch time for different types of workload and increased number of CPUs. MR has shown the best total completion times for AuverGrid workload traces whilst MH and MHR have shown the best total completion times for LCG1 and synthetic workload traces. The RR and MDQ have shown the best average response time for different types of workloads, with the exception of AuverGrid workload traces. RR has produced worst average response times for AuverGrid workload traces because of the nature of those traces. For job execution using AuverGrid workload traces, too many jobs arrived during very short time-interval, as a result queue size has become very large. RR could not respond to user jobs for long time, and delay occurred which has produced longer average response times. The MH, MHR and MDQ scheduling algorithms have shown the best performance under the increasing number of CPUs, and relative performance under different workloads (synthetic, LCG1 and AuverGrid) conditions.

Section 6.6.5.7 represents that performance and efficiency of scheduling algorithms, i.e., RR, MH and MDQ are also dependent on the value of time quantum. These three algorithms have shown decaying in performance under the increasing values of time quantum from ‘50’ to ‘5000’ for ‘10%’ workload of LCG1. RR has shown the best performance parameters at ‘50’ seconds of time quantum, and shown the worst performance results at ‘5000’ seconds of time quantum. MH has shown the best performance for each performance parameters compared to RR and MDQ, but represented poor performance measures under the increasing values of time quantum.

The performance of proposed job scheduling algorithms (MH, MHR and MDQ) are independent of the type of the workload, the workload sizes and the number of CPUs used in the experiments. These proposed scheduling algorithms markedly outperform than other grid scheduling algorithms. A significant improvement has achieved in all of the performance parameters. They are adaptive to grid dynamics, and possess a high degree of performance, efficiency and scalability. It has been demonstrated and concluded that MH and MHR are better scheduling algorithms from system perspective while MDQ and RR are better choices from user perspective.

6.7 Chapter Summary

This chapter has presented comparative performance analysis of proposed scheduling algorithms compared to existing approaches. These algorithms have been evaluated using simulator on an experimental grid using synthetic and real workload traces (i.e., LCG1 and AuverGrid) under dynamic scheduling environment.

Experimental results shows that the MH and MHR scheduling algorithms have shown the best average waiting times, average turnaround times, average slowdown times, total completion times and maximum job stretch times compared to other scheduling algorithms under different workload conditions. MR also has shown the best total completion times for AuverGrid workload traces. MHR can show its better performance than MH due to its non-affected performance by the value of a fixed time quantum. Experimental results also exhibit that RR and MDQ have shown the best average response times compared to other approaches.

It has been experimentally concluded that MH and MHR are scheduling policies from the system point of view. RR and MDQ works well from the user perspective due to its short average response times. Furthermore, it is also concluded that the MH, MHR and MDQ are scalable, i.e. the relationship between each performance measure (e.g. average waiting time) and the workload size is very nearly linear.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

7.1 Chapter Overview

This chapter focuses on the main contributions of this thesis, limitations in this work, and possible future direction.

The structure of this chapter is as follows: Section 7.2 describes the main contribution of this thesis. Section 7.3 outlines the limitations and possible future work on grid scheduling problems addressed in this thesis, as well as possible alternatives to improve them.

7.2 Research Contributions

This thesis has presented a grid scheduling model. In addition, new algorithms for efficient and effective resource allocation and job scheduling on computational grid are proposed in this thesis. Additionally, this work has presented three simulators to evaluate the efficiency of proposed algorithms compared with existing approaches. Relationship between these simulators is shown in Figure 7.1.

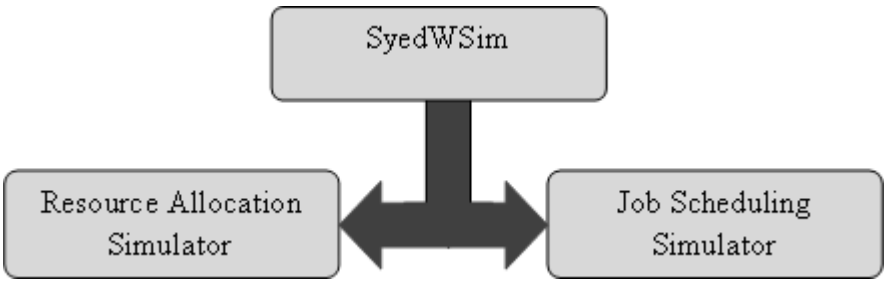

Figure 7.1 Relationship between simulators

The web-based simulator (i.e., *SyedWSim*) is for statistical analysis of grid workload traces. *SyedWSim* has been used to characterize the real workload traces (i.e., LCG1 and AuverGrid) from user, groups and grid system perspective. The real grid workloads traces have shown rich correlation structure and scaling behavior, and have been used in this work for performance evaluation of resource allocation and job scheduling algorithms.

Two simulators namely '*Resource allocation simulator'* and '*Job scheduling simulator'* have been developed to evaluate the efficiency, performance and scalability of proposed and existing algorithms for resource allocation and job scheduling respectively. These simulators have used the synthetic and real workload traces as input for evaluation purposes. *Resource allocation simulator* has produced the comprehensive simulation for each resource allocation method. This simulator has shown the mapping of job(s) to processor(s) at each step during resource allocation activity. *Job scheduling simulator* has facilitated this research by evaluating the efficiency and performance of each scheduling algorithm on an experimental grid using a large number of synthetic and real workload traces in dynamic grid scheduling environment. This simulator has the tendency to communicate with grid nodes physically by message passing API (i.e., MPJ-express) and distribute the workload among computing nodes (i.e., slave processors), and perform the computation using each scheduling algorithm at each node. Finally it would the different performance parameters (i.e., waiting times, turnaround times etc) from the participating nodes and then compute the average performance measures (i.e., average waiting time, average turnaround time etc) at system level.

In this work, baseline and proposed approaches considered for resource allocation are listed as follows:

Baseline Approaches

- Min-Min algorithm
- Max-Min algorithm
- Vogel Approximation method (VAM)

Proposed Method

- Modified Least Cost Method (MLCM)

- First Come First Served (FCFS)
- Divisible Load Theory method (DLT)

This thesis has proposed MLCM for allocation of tasks to computing resources efficiently. The performance, efficiency and scalability of MLCM have been evaluated and compared with other baseline approaches using simulation for different grid resource allocation scenarios that are composed of synthetic and real workload traces (i.e., LCG1). Experiments have been conducted under the increased number of CPUs from '5' to '128' by varying the size of workload and by varying the computing capabilities of each processor in the simulation setup. Experimental results have shown that MLCM performs well and produces the lower computational cost (in terms of time) in comparison to other resource allocation methods for a variety of resource allocation scenarios. Out of all the algorithms, MLCM has shown the shorter computational cost by increasing the number of CPUs and relative performance under increased workload. Experimental results have also depicted that Min-Min has shown slightly shorter computational cost than MLCM in one particular case where the grid environment mostly consisted of processors having higher computing power. In summary, in eight out of eleven scenarios, MLCM is superior to Min-Min, in two scenarios they are equivalent, and in one scenario, Min-Min is superior. As a result, MLCM has shown best computational cost and supports scalability

In this work, baseline and proposed approaches considered for jobs scheduling are listed below:

Baseline Approaches

- First Come First Served (FCFS)
- Shorted Process Next (SPN)
- Longest Job First (LJF)
- Priority(P)
- Round Robin (RR)
- Proportional Local Round Robin (PLRR)

Proposed Scheduling Algorithms

- Multilevel Hybrid scheduling algorithms (MH)
- Multilevel Dual Queue Scheduling algorithms (MDQ)
- Dynamic Multilevel Hybrid Scheduling Algorithm using Median(MHM)
- Dynamic Multilevel Hybrid Scheduling Algorithm using square root(MHR)

- Self Adjustment Round Robin (SARR)
- Intelligent Time slice for Round Robin (NIR)
- Round Robin Priority (NRR)
- Multilevel CPU Scheduling algorithm(MR)
- Shortest Remaining Burst Round Robin (SRBRR)
- Dynamic Multilevel Dual Queue Scheduling Algorithm using Median (MDQM)
- Dynamic Multilevel Dual Queue Scheduling Algorithm using Square root (MDQR)

Experiments have been performed to evaluate the efficiency, performance and scalability of proposed and existing scheduling algorithms using simulation on an experimental grid for synthetic and real grid workload traces under dynamic scheduling environment. Performance evaluation criteria was based on the following six metrics: average waiting time, average turnaround time, average response time, average bounded time, flow time and job stretch time. One out of six metrics i.e., average response time was required to minimize from user perspective whilst other five were needed to minimize from system perspective. These performance metrics have been computed for each scheduling algorithm under the increasing workload and varying the number of CPUs successively; and using variety of workloads (i.e., synthetic, LCG1, AuverGrid).

The experimental results based on the performance evaluation criteria have shown that MH and MHR have shown the best average waiting time, average turnaround time, average slowdown time, total completion time and job stretch time. The RR and MDQ have shown the best average response time for different types of workloads, with the exception of AuverGrid workload traces. RR has produced worst average response times for AuverGrid workload traces because too many jobs arrived during very short time-interval; as a result queue size has become very large. RR could not respond to user jobs for long time and delay occurred which has produced longer average response times. The MH, MHR and MDQ scheduling algorithms have shown the best performance under the increasing number of CPUs and relative performance under different workloads (synthetic, LCG1 and AuverGrid) conditions.

The performance of proposed job scheduling algorithms (MH, MHR and MDQ) are independent of the type of the workload, the workload sizes and the number of CPUs used in the experiments. These proposed algorithms are adaptive to grid dynamics and also possess a high degree of performance, efficiency and scalability. It has been demonstrated and concluded that MH and MHR are better scheduling algorithms from system perspective while MDQ and RR are better choices from user perspective.

7.3 Research Achievements

The main achievements from this research are mentioned as follows:

1. Grid scheduling model is proposed for designing and evaluation of grid scheduling algorithms.
2. A web based simulator (i.e.; SyedWSim) has been designed and developed for analysis and characterization of grid workload traces.
3. A new grid resource allocation method (i.e.; MLCM) is proposed which has shown efficient utilization of resources and lowered the cost of computation.
4. MLCM has shown the optimal computational cost compared to other grid resource allocation methods for different resource allocation scenarios. It has experimentally proven that MLCM is a promising technique to use in grid environment, when dealing with tasks allocation.
5. New job scheduling algorithms including Multilevel Hybrid Scheduling (MH), Multilevel Dual Queue Scheduling (MDQ), Dynamic Multilevel Hybrid Scheduling (i.e., MHM and MHR) and Dynamic Multilevel Dual Queue Scheduling (i.e., MDQM and MDQR) are proposed for efficient and fair job execution
6. The proposed algorithms have been evaluated by comparing with other well known scheduling algorithms for various scheduling performance metrics using synthetic and real workload traces in dynamic grid scheduling environment. Experimental results demonstrated that MH, MHR and MDQ have shown better performance, efficiency and scalability. MH and MHR have

shown better performance from system perspective whilst MDQ has produced better performance from user perspective.

7.4 Limitations and Future work

In proposed scheduling algorithms the task dependencies have not been considered. There are many constraints which can be included in the scheduling process, but as an immediate future work, task dependencies will be considered in the grid scheduling algorithms.

The developed algorithms can be deployed in real time environment to attain high computing power for processing of complex scientific and engineering problems. Campus wide grid can be one of its potential applications. Engineers and scientist would be able to execute their high computing demanding applications on campus wide grid in an efficient and effective way.

Security is also another significant aspect in design of grid scheduling model. Security can be seen from two perspectives. Firstly; tasks could be allocated to secure nodes within the grid. Secondly; the tasks running at the resource will not be able to see or access other data in the other specific node of grid. The existing security approaches are actually practiced at various levels of grid systems and work independently of the grid schedulers. It is also really worth challenging to integrate the security/ trust level as one of the aims in the grid scheduling model. Security can also be one of the considerations in the enhancement of grid scheduling algorithms.

Agent technology is suitable for a computational grid because of the dynamic, heterogeneous, and autonomous nature of the grid. At present, there is need to design and develop robust grid scheduling framework using agent technology. Proposed scheduling algorithms can be integrated with this framework to provide robust and reliable solution. Later on, this framework can be deployed in a real time environment to attain high computational power for processing of scientific and engineering applications in a robust fashion.

Cloud computing is another important advancement in the high performance computing domain. Although there is no precise definition of cloud computing, most cloud computing infrastructure, by making usage of virtualization approaches, allow users to set-up customized computing environments on demand. Amazon EC2 and Google App Engine are two of the more important cloud computing services with the former offering cloud infrastructure (hardware) and the later offering platform (software). Although virtualization approaches make the resources appear homogenous, the actual resources remain distributed. Therefore, cloud computing can be viewed as another platform on which propose scheduling algorithms can be utilized to run applications.

REFERENCES

[1] I. F. a. C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, 1999.

[2] *Top500 Supercomputing*. Available: http://www.top500.org/ [Jul. 02, 2011].

[3] *Supercomputer "K computer" Takes First Place in World*. Available: http://www.fujitsu.com/global/news/pr/archives/month/2011/20110620-02.html [Jul. 02, 2011].

[4] S. M. Anthony Mayer, Nathalie Furmento,William Lee, Murtaza Gulamali,Steven Newhouse,John Darlington, "Workflow Expression: Comparison of Spatial and Temporal Approaches," in *Proceedings of the Workflow in Grid Systems Workshop, GGF-10*, Berlin, 2004.

[5] Dbrain. (2011). Available: http://www.dementia.com.my [June. 12, 2010].

[6] Mayoclinic. (2011). Available: http://www.mayoclinic.com/ [May. 04, 2010].

[7] Neurogrid. *Economic and On Demand "Brain Activity Analysis" on the World Wide Grid Using Nimrod-G and Gridbus Technologies*. Available: http://www.gridbus.org/neurogrid/ [May. 02, 2010].

[8] PPDG, "Particle Physics Data Grid."

[9] Geodise. *Geodise: Aerospace Design Optimisation*. Available: www.geodise.org [May. 02, 2010].

[10] MIMOS. *Grid Computing -The Technology*. Available: http://www.mimos.my/technology-thrust-areas/grid-computing/about-us4/ [Aug. 15, 2009].

[11] R. Buyya, D. Abramson, and J. Giddy, "Nimrod/G: an architecture for a resource management and scheduling system in a global computational grid," 2000, pp. 283-289 vol.1.

[12] P. Huang, H. Peng, P. Lin, and X. Li, "Static strategy and dynamic adjustment: An effective method for Grid task scheduling," *Future Generation Computer Systems*, vol. 25, pp. 884-892, 2009.

[13] J. Chin, M. Harvey, S. Jha, and P. Coveney, "Scientific Grid Computing: The First Generation," *Computing in Science and Engineering*, vol. 7, pp. 24-32, 2005.

[14] Berstis, "Fundamentals of Grid Computing," in *IBM RedBooks Paper*, ed, 2002.

[15] K. Krauter, Buyya, R., Maheswaran, M., "A taxonomy and survey of grid resource management systems for distributed computing," *Softw. Pract. Exper.*, vol. 32, pp. 135-164, 2002.

[16] L. Chunlin, Z. J. Xiu, and L. Layuan, "Resource scheduling with conflicting objectives in grid environments: Model and evaluation," *Journal of Network and Computer Applications*, vol. 32, pp. 760-769, 2009.

[17] E. Shmueli and D. G. Feitelson, "Backfilling with lookahead to optimize the packing of parallel jobs," *J. Parallel Distrib. Comput.*, vol. 65, pp. 1090-1107, 2005.

[18] F. Xhafa and A. Abraham, "Computational models and heuristic methods for Grid scheduling problems," *Future Gener. Comput. Syst.*, vol. 26, pp. 608-621, 2010.

[19] R. Buyya, D. Abramson, and J. Giddy, "Grid Resource Management, Scheduling and Computational Economy," ed.

[20] D. Ouelhadj and S. Petrovic, "A survey of dynamic scheduling in manufacturing systems," *J. of Scheduling*, vol. 12, pp. 417-431, 2009.

[21] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *Int. J. High Perform. Comput. Appl.*, vol. 15, pp. 200-222, 2001.

[22] X.-H. Sun and M. Wu, "Quality of Service of Grid Computing: Resource Sharing," in *Grid and Cooperative Computing, 2007. GCC 2007. Sixth International Conference on*, 2007, pp. 395-402.

[23] K. Krauter, K. Klaus, and M. It. (2001). *A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing*. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.15.5054 [Sept. 11, 2008].

[24] L. Peng, S. See, Y. Jiang, J. Song, A. Stoelwinder, and H. Neo, "Performance evaluation in computational grid environments," 2004, pp. 54-62.

[25] J. Yu and R. Buyya, "A Taxonomy of Workflow Management Systems for Grid Computing," *Journal of Grid Computing*, vol. 3, pp. 171-200, 2005.

[26] D. Fernandez-Baca, "Allocating modules to processors in a distributed system," *Software Engineering, IEEE Transactions on*, vol. 15, pp. 1427-1436, 1989.

[27] P. W. Ramin Yahyapour, "Grid Scheduling Use Cases," 2006.

[28] V. T'Kindt and J.-C. Billaut, *Multicriteria Scheduling: Theory, Models and Algorithms*: Springer, 2006.

[29] F. D. a. S. G. Akl. (2006, Scheduling Algorithms for Grid Computing: State of the Art and Open Problems.

[30] CoreGRID. *CoreGRID*. Available: http://www.coregrid.net/ [Jul. 05, 2009].

[31] CoreGRID. (2007), Review of the Performance Evaluation and Benchmarking of Grid Scheduling Systems. Available: http://www.coregrid.net/mambo/images/stories/Deliverables/d.rms.05.pdf [Jul. 05, 2009].

[32] F. Xhafa, L. Barolli, J. A. Gonzalez, and P. Jura, "A Static Benchmarking for Grid Scheduling Problems," presented at the Proceedings of the 2009 International Conference on Advanced Information Networking and Applications Workshops, 2009.

[33] S. D. Anoep, "Trace-based Performance Analysis of Scheduling Bags of Tasks in Grids," 2009.

[34] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, and D. Epema, "The Grid Workloads Archive," *Future Generation Computer Systems*, vol. 24, pp. 672-686, 2008.

[35] D. KALINSKY. Who’s Afraid of Asymmetric Multiprocessing? *RTC*. Available: http://www.rtcmagazine.com/articles/view/100404 [Sept. 17, 2010].

[36] L. Lamport, "How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs," *IEEE Trans. Comput.*, vol. 28, pp. 690-691, 1979.

[37] W. W. Gropp and E. L. Lusk, "A taxonomy of programming models for symmetric multiprocessors and SMP clusters," in *Programming Models for Massively Parallel Computers, 1995*, 1995, pp. 2-7.

[38] L. Dagum and R. Menon, "OpenMP: an industry standard API for shared-memory programming," *Computational Science & Engineering, IEEE,* vol. 5, pp. 46-55, 1998.

[39] N. Robertson and A. Rendell, "OpenMP and NUMA Architectures I: Investigating Memory Placement on the SGI Origin 3000," in *Computational Science — ICCS 2003*. vol. 2660, P. Sloot, *et al.*, Eds., ed: Springer Berlin / Heidelberg, 2003, pp. 722-722.

[40] J. Nieplocha, R. J. Harrison, and R. J. Littlefield, "Global arrays: a nonuniform memory access programming model for high-performance computers," *J. Supercomput.,* vol. 10, pp. 169-189, 1996.

[41] R. Buyya, *High Performance Cluster Computing: Architectures and Systems*: Prentice Hall PTR, Upper Saddle River, NJ, USA,, 1999.

[42] R. Buyya, "High Performance Cluster Computing: Programming and Applications," vol. 2, 1999.

[43] Staff, "Using MPI-Portable Parallel Programming with the Message-Passing Interface, by William Gropp," *Sci. Program.,* vol. 5, pp. 275-276, 1996.

[44] A. B. Al Geist, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam, "PVM: Parallel virtual machine: A users' guide and tutorial for networked parallel computing : By Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek and Vaidy Sunderam. MIT Press, Cambridge, MA. (1994). 279 pages. $19.95," *Computers & Mathematics with Applications,* vol. 30, p. 122, 1995.

[45] A. S. T. a. M. V. Steen, *Distributed Systems: Principles and Paradigms*: Prentice-Hall, 2001.

[46] I. Foster, "What is the Grid? - a three point checklist," *GRIDtoday,* vol. 1, 2002.

[47] A. Iosup, M. Jan, O. Sonmez, and D. H. J. Epema, "On the dynamic resource availability in grids," presented at the Proceedings of the 8th IEEE/ACM International Conference on Grid Computing, 2007.

[48] D. Lim, Y.-S. Ong, Y. Jin, B. Sendhoff, and B.-S. Lee, "Efficient Hierarchical Parallel Genetic Algorithms using Grid computing," *Future Gener. Comput. Syst.,* vol. 23, pp. 658-670, 2007.

[49] G. Johnson, "All Science is Computer Science," in *The New York Times*, ed, 2001.

[50] S. Jang and J. Lee, "Predictive Grid Process Scheduling Model in Computational Grid," in *Advanced Web and Network Technologies, and Applications*. vol. 3842, H. Shen, *et al.*, Eds., ed: Springer Berlin / Heidelberg, 2006, pp. 525-533.

[51] H. B. Newman, M. H. Ellisman, and J. A. Orcutt, "Data-intensive e-science frontier research," *Commun. ACM*, vol. 46, pp. 68-77, 2003.

[52] F. Xhafa, C. Paniagua, L. Barolli, and S. Caballe, "A parallel grid based implementation for real time processing of event log data of collaborative applications," *Int. J. Web Grid Serv.*, vol. 6, pp. 124-140, 2010.

[53] M. D. Beynon, A. Sussman, T. Kure, and J. Saltz, "Performance Optimization for Data Intensive Grid Applications," presented at the Proceedings of the Third Annual International Workshop on Active Middleware Services, 2001.

[54] H. Casanova and J. Dongarra, "Network enabled solvers for scientific computing using the NetSolve system," in *Algorithms and Architectures for Parallel Processing, 1997. ICAPP 97., 1997 3rd International Conference on*, 1997, pp. 17-33.

[55] J. P. Goux, S. Kulkarni, J. Linderoth, and M. Yoder, "An enabling framework for master-worker applications on the Computational Grid," in *High-Performance Distributed Computing, 2000. Proceedings. The Ninth International Symposium on*, 2000, pp. 43-50.

[56] J. Linderoth and S. Wright, "Decomposition Algorithms for Stochastic Programming on a Computational Grid," *Comput. Optim. Appl.*, vol. 24, pp. 207-250, 2003.

[57] F. Xhafa and A. Abraham, "Meta-heuristics for Grid Scheduling Problems," in *Metaheuristics for Scheduling in Distributed Computing Environments*. vol. 146, F. Xhafa and A. Abraham, Eds., ed: Springer Berlin / Heidelberg, 2008, pp. 1-37.

[58] V. B. Ferrira, J. Armstrong, M. Kendzlerski, Andreas Neukoetter, M. Takagl, R. Bing-Wo, A. Amir, R. Murakawa, O. Hernandez, J. Magowan, and N. Bleberstein. (2003, Introduction the Grid Computing with Globus. Available: http://www.redbooks.ibm.com/abstracts/sg246895.html [Oct. 21, 2009].

[59] B. F. Z. Xu, W. Li, *Grid Computing Technology*: China Electronics Press, 2004.

[60] L. Smarr and C. E. Catlett, "Metacomputing," *Commun. ACM,* vol. 35, pp. 44-52, 1992.

[61] Z. Lichen, "Scheduling algorithm for real-time applications in grid environment," in *Systems, Man and Cybernetics, 2002 IEEE International Conference on*, 2002, p. 6 pp. vol.5.

[62] H. Kurdi, M. Li, and H. Al-Raweshidy, "A Classification of Emerging and Traditional Grid Systems," *IEEE Distributed Systems Online,* vol. 9, p. 1, 2008.

[63] K. Amin, G. von Laszewski, and A. R. Mikler, "Grid computing for the masses: an overview," in *Grid and Cooperative Computing. Second International Workshop (GCC 2003)*, Shanghai, China, pp. 464-73 BN - 3 540 21988 9.

[64] Y. Deng and F. Wang, "Opportunities and challenges of storage grid enabled by grid service," *SIGOPS Oper. Syst. Rev.,* vol. 41, pp. 79-82, 2007.

[65] J. D. a. J. Skelton. *UC Berkeley Storage Area Network (SAN)*. Available: http://inews.berkeley.edu/bcc/Spring2005/san.html

[66] X. Jin, J. Liu, and Z. Yang, "Modeling Agent-Based Task Handling in a Peer-to-Peer Grid," presented at the Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology, 2004.

[67] A. Iamnitchi, M. Ripeanu, and I. T. Foster, "Locating Data in (Small-World?) Peer-to-Peer Scientific Collaborations," presented at the Revised Papers from the First International Workshop on Peer-to-Peer Systems, 2002.

[68] Escience-grid. *http://www.escience-grid.org.uk/* [Mar. 07, 2010].

[69] P. Strong. (2005). *Enterprise Grid Computing*. Available: http://queue.acm.org/detail.cfm?id=1080877 [Mar. 07, 2010].

[70] SZTAKI. *Desktop Grid*. Available: http://www.desktopgrid.hu/ [Mar. 07, 2010].

[71] D. Kondo, M. Taufer, C. L. Brooks, H. Casanova, and A. A. Chien, "Characterizing and evaluating desktop grids: an empirical study," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, 2004, p. 26.

[72] V. Berstis, "IBM RedBook; Fundamentals of Grid Computing," 2002.

[73] J. M. Schopf, "Ten actions when Grid scheduling: the user as a Grid scheduler," in *Grid resource management*, N. Jarek, *et al.*, Eds., ed: Kluwer Academic Publishers, 2004, pp. 15-23.

[74] J. a. S. Nabrzyski, Jennifer M. and Weglarz, Jan, "Grid resource management: state of the art and future trends," 2004.

[75] Y. Bartal, A. Fiat, H. Karloff, and R. Vohra, "New algorithms for an ancient scheduling problem," presented at the Proceedings of the twenty-fourth annual ACM symposium on Theory of computing, Victoria, British Columbia, Canada, 1992.

[76] CoreGRID. (2006). Proposal of Multi-level Scheduling Models. Available: http://www.coregrid.net/mambo/images/stories/Deliverables/d.rms.04.pdf [May. 11, 2010].

[77] K. Czajkowski, I. Foster, and C. Kesselman, "Resource co-allocation in computational grids," in *High Performance Distributed Computing, 1999. Proceedings. The Eighth International Symposium on*, 1999, pp. 219-228.

[78] H. Yu, X. Bai, and D. Marinescu, "Workflow management and resource discovery for an intelligent grid," *Parallel Computing*, vol. 31, pp. 797-811, 2005.

[79] J. Yu and R. Buyya, "A taxonomy of scientific workflow systems for grid computing," *SIGMOD Rec.*, vol. 34, pp. 44-49, 2005.

[80] I. M. J. Palmer "Optimal Tree Structures for Large-Scale Grids," in *Proceedings of the UK e-Science*, 2005.

[81] K. Christodoulopoulos, V. Sourlas, I. Mpakolas, and E. Varvarigos, "A comparison of centralized and distributed meta-scheduling architectures for computation and communication tasks in Grid networks," *Computer Communications*, vol. 32, pp. 1172-1184, 2009.

[82] J. Abawajy, "Job Scheduling Policy for High Throughput Grid Computing," in *Distributed and Parallel Computing*. vol. 3719, M. Hobbs, *et al.*, Eds., ed: Springer Berlin / Heidelberg, 2005, pp. 184-192.

[83] L. Liang-Teh, L. Chin-Hsiian, and C. Hung-Yuan, "An Adaptive Task Scheduling System for Grid Computing," in *Computer and Information*

Technology, 2006. CIT '06. The Sixth IEEE International Conference on, 2006, pp. 57-57.

[84] H. Casanova, M. Kim, J. S. Plank, and J. Dongarra, "Adaptive Scheduling for Task Farming with Grid Middleware," presented at the Proceedings of the 5th International Euro-Par Conference on Parallel Processing, 1999.

[85] A. Othman, P. Dew, K. Djemame, and I. Gourlay, "Adaptive grid resource brokering," in *Cluster Computing, 2002. Proceedings. 2002 IEEE International Conference on*, 2003, pp. 172-179.

[86] E. Huedo, R. S. Montero, and I. M. Llorente, "Experiences on adaptive grid scheduling of parameter sweep applications," in *Parallel, Distributed and Network-Based Processing, 2004. Proceedings. 12th Euromicro Conference on*, 2004, pp. 28-33.

[87] S. Ali, H. J. Siegel, M. Maheswaran, and D. Hensgen, "Task execution time modeling for heterogeneous computing systems," in *Heterogeneous Computing Workshop, 2000. (HCW 2000) Proceedings. 9th*, 2000, pp. 185-199.

[88] S. Hotovy, "Workload Evolution on the Cornell Theory Center IBM SP2," presented at the Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing, 1996.

[89] N. Fujimoto and K. Hagihara, "Near-optimal dynamic task scheduling of precedence constrained coarse-grained tasks onto a computational grid," in *Parallel and Distributed Computing, 2003. Proceedings. Second International Symposium on*, 2003, pp. 80-87.

[90] Y. C. Lee and A. Y. Zomaya, "Practical Scheduling of Bag-of-Tasks Applications on Grids with Dynamic Resilience," *Computers, IEEE Transactions on*, vol. 56, pp. 815-825, 2007.

[91] C. Li, "Competitive proportional resource allocation policy for computational grid," *Future Generation Computer Systems*, vol. 20, pp. 1041-1054, 2004.

[92] B. Rashmi and D. P. Agrawal, "Improving scheduling of tasks in a heterogeneous environment," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 15, pp. 107-118, 2004.

[93] Z. Shi and J. J. Dongarra, "Scheduling workflow applications on processors with different capabilities," *Future Gener. Comput. Syst.*, vol. 22, pp. 665-675, 2006.

[94] S. Ranaweera and D. P. Agrawal, "A task duplication based scheduling algorithm for heterogeneous systems," in *Parallel and Distributed Processing Symposium, 2000. IPDPS 2000. Proceedings. 14th International*, 2000, pp. 445-450.

[95] B. Ravindran, P. Li, and T. Hegazy, "Proactive resource allocation for asynchronous real-time distributed systems in the presence of processor failures," *J. Parallel Distrib. Comput.*, vol. 63, pp. 1219-1242, 2003.

[96] C. Zhu, Z. Liu, W. Zhang, W. Xiao, Z. Xu, and D. Yang, "Decentralized Grid Resource Discovery Based on Resource Information Community," *Journal of Grid Computing*, vol. 2, pp. 261-277, 2004.

[97] A. Galstyan, K. Czajkowski, and K. Lerman, "Resource Allocation in the Grid Using Reinforcement Learning," presented at the Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 3, New York, New York, 2004.

[98] F. de O. Lucchese, E. Huerta Yero, F. Sambatti, and M. Henriques, "An Adaptive Scheduler for Grids," *Journal of Grid Computing*, vol. 4, pp. 1-17, 2006.

[99] R. Armstrong, D. Hensgen, and T. Kidd, "The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions," in *Heterogeneous Computing Workshop, 1998. (HCW 98) Proceedings. 1998 Seventh*, 1998, pp. 79-87.

[100] R. F. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J. D. Lima, F. Mirabile, L. Moore, B. Rust, and H. J. Siegel, "Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet," in *Heterogeneous Computing Workshop, 1998. (HCW 98) Proceedings. 1998 Seventh*, 1998, pp. 184-199.

[101] M. Maheswaran, S. Ali, H. J. Siegal, D. Hensgen, and R. F. Freund, "Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems," in *Heterogeneous Computing Workshop, 1999. (HCW '99) Proceedings. Eighth*, 1999, pp. 30-44.

[102] J. Yu, R. Buyya, and K. Ramamohanarao, "Workflow Scheduling Algorithms for Grid Computing," in *Metaheuristics for Scheduling in Distributed Computing Environments*. vol. 146, F. Xhafa and A. Abraham, Eds., ed: Springer Berlin / Heidelberg, 2008, pp. 173-214.

[103] T. D. Braun, H. J. Siegel, N. Beck, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *J. Parallel Distrib. Comput.*, vol. 61, pp. 810-837, 2001.

[104] Y. Xiaogao and Y. Xiaopeng, "A New Grid Computation-Based Min-Min Algorithm," in *Fuzzy Systems and Knowledge Discovery, 2009. FSKD '09. Sixth International Conference on*, 2009, pp. 43-45.

[105] K. Etminani and M. Naghibzadeh, "A Min-Min Max-Min selective algorihtm for grid task scheduling," in *Internet, 2007. ICI 2007. 3rd IEEE/IFIP International Conference in Central Asia on*, 2007, pp. 1-7.

[106] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman, "Heuristics for scheduling parameter sweep applications in grid environments," in *Heterogeneous Computing Workshop, 2000. (HCW 2000) Proceedings. 9th*, 2000, pp. 349-363.

[107] H. Topcuouglu, S. Hariri, and M.-y. Wu, "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, pp. 260-274, 2002.

[108] G. a. C. Murugesan, C., "An Economical Model for Optimal Distribution of Loads for Grid Applications," *International Journal of Computer and Network Security*, 2009.

[109] H. A. Taha. *TORA Optimization System*. Available: http://prenhall.com/taha/ [Dec. 15, 2009].

[110] H. A. Taha, *Operations Research: An Introduction*, 8th ed.: Prentice Hall, Inc., 2007.

[111] Y. Zhang, H. Franke, J. E. Moreira, and A. Sivasubramaniam, "Improving parallel job scheduling by combining gang scheduling and backfilling techniques," in *Parallel and Distributed Processing Symposium, 2000. IPDPS 2000. Proceedings. 14th International*, 2000, pp. 133-142.

[112] M. Dhodhi, "An Integrated Technique for Task Matching and Scheduling onto Distributed Heterogeneous Computing Systems," *Journal of Parallel and Distributed Computing,* vol. 62, pp. 1338-1361, 2002.

[113] S.-Y. Lee and C.-H. Cho, "Load Balancing for Minimizing Execution Time of a Target Job on a Network of Heterogeneous Workstations," presented at the Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing, 2000.

[114] S. Wiriyaprasit and V. Muangsin, "The impact of local priority policies on grid scheduling performance and an adaptive policy-based grid scheduling algorithm," in *High Performance Computing and Grid in Asia Pacific Region, 2004. Proceedings. Seventh International Conference on*, 2004, pp. 343-346.

[115] Y. M. Teo, X. Wang, and J. P. Gozali, "A Compensation-based Scheduling Scheme for Grid Computing," presented at the Proceedings of the High Performance Computing and Grid in Asia Pacific Region, Seventh International Conference, 2004.

[116] B. G. Lawson, E. Smirni, and D. Puiu, "Self-Adapting Backfilling Scheduling for Parallel Systems," presented at the Proceedings of the 2002 International Conference on Parallel Processing, 2002.

[117] D. Tsafrir, Y. Etsion, and D. G. Feitelson, "Backfilling Using System-Generated Predictions Rather than User Runtime Estimates," *Parallel and Distributed Systems, IEEE Transactions on,* vol. 18, pp. 789-803, 2007.

[118] K. Li, "Job scheduling and processor allocation for grid computing on metacomputers," *Journal of Parallel and Distributed Computing,* vol. 65, pp. 1406-1418, 2005.

[119] W. Stallings, *Operating Systems Internals and Design Principles*: Prentice Hall, 2004.

[120] H. Li and R. Buyya, "Model-Driven Simulation of Grid Scheduling Strategies," presented at the Proceedings of the Third IEEE International Conference on e-Science and Grid Computing, 2007.

[121] R. J. Matarneh, "Self-Adjustment Time Quantum in Round Robin Algorithm Depending on Burst Time of the Now Running Processes," *American Journal of Applied Sciences,* 2009.

[122] R. R. Yaashuwanth.C, "Design of Real Time scheduler simulator and Development of Modified Round Robin Archetectue," *IJCSNS International Journal of Computer Science and Network Security*, 2010.

[123] R. R. Yaashuwanth.C, "Intelligent Time Slice for Round Robin in Real Time Operating System," *IJRRAS*, vol. 2, 2010.

[124] M. M. R. a. M. N. Akhtar, "A New Mutilevel CPU Scheduling Algorithm," *Journal of Applied Sciences* vol. 6, 2009.

[125] H. S. B. Rakesh Mohanty, Khusbu Patwari, Manas Ranjan Das, Monisha Dash, Sudhashree, "Design and Performance Evaluation of a New Proposed Shortest Remaining Burst Round Robin (SRBRR) Scheduling Algorithm," in *ISCET 2010*.

[126] D. Lifka, "The ANL/IBM SP scheduling system," in *Job Scheduling Strategies for Parallel Processing*. vol. 949, D. Feitelson and L. Rudolph, Eds., ed: Springer Berlin / Heidelberg, 1995, pp. 295-303.

[127] J. Skovira, W. Chan, H. Zhou, and D. Lifka, "The EASY — LoadLeveler API project," in *Job Scheduling Strategies for Parallel Processing*. vol. 1162, D. Feitelson and L. Rudolph, Eds., ed: Springer Berlin / Heidelberg, 1996, pp. 41-47.

[128] D. G. Feitelson, "A Survey of Scheduling in Multiprogrammed Parallel Systems," IBM T.J.Watson Research Center, Yorktown Heights, NY1995.

[129] A. W. Mu'alem and D. G. Feitelson, "Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 12, pp. 529-543, 2001.

[130] S. S. Rawat and L. Rajamani, "Experiments with CPU Scheduling Algorithm on a Computational Grid," in *Advance Computing Conference, 2009. IACC 2009. IEEE International*, 2009, pp. 71-75.

[131] R. Sharma, V. K. Soni, and M. K. Mishra, "An improved resource scheduling approach using Job Grouping strategy in grid computing," in *Educational and Network Technology (ICENT), 2010 International Conference on*, 2010, pp. 94-96.

[132] A. Tchernykh, D. Trystram, C. Brizuela, and I. Scherson, "Idle regulation in non-clairvoyant scheduling of parallel jobs," *Discrete Appl. Math.*, vol. 157, pp. 364-376, 2009.

[133] L. Hui and R. Buyya, "Model-Driven Simulation of Grid Scheduling Strategies," in *e-Science and Grid Computing, IEEE International Conference on*, 2007, pp. 287-294.

[134] D. Feitelson and L. Rudolph, "Metrics and benchmarking for parallel job scheduling," in *Job Scheduling Strategies for Parallel Processing*. vol. 1459, D. Feitelson and L. Rudolph, Eds., ed: Springer Berlin / Heidelberg, 1998, pp. 1-24.

[135] M. Calzarossa and G. Serazzi, "Workload characterization: a survey," *Proceedings of the IEEE*, vol. 81, pp. 1136-1150, 1993.

[136] S. Chapin, W. Cirne, D. Feitelson, J. Jones, S. Leutenegger, U. Schwiegelshohn, W. Smith, and D. Talby, "Benchmarks and Standards for the Evaluation of Parallel Job Schedulers," in *Job Scheduling Strategies for Parallel Processing*. vol. 1659, D. Feitelson and L. Rudolph, Eds., ed: Springer Berlin / Heidelberg, 1999, pp. 67-90.

[137] C. Ernemann, B. Song, and R. Yahyapour, "Scaling of Workload Traces," in *Job Scheduling Strategies for Parallel Processing*. vol. 2862, D. Feitelson, *et al.*, Eds., ed: Springer Berlin / Heidelberg, 2003, pp. 166-182.

[138] D. G. Feitelson, "Metric and workload effects on computer systems evaluation," *Computer*, vol. 36, pp. 18-25, 2003.

[139] E. Frachtenberg and D. Feitelson, "Pitfalls in Parallel Job Scheduling Evaluation," in *Job Scheduling Strategies for Parallel Processing*. vol. 3834, D. Feitelson, *et al.*, Eds., ed: Springer Berlin / Heidelberg, 2005, pp. 257-282.

[140] D. Feitelson, "Workload Modeling for Performance Evaluation," in *Performance Evaluation of Complex Systems: Techniques and Tools*. vol. 2459, M. Calzarossa and S. Tucci, Eds., ed: Springer Berlin / Heidelberg, 2002, pp. 114-141.

[141] S.-H. Chiang and M. K. Vernon, "Characteristics of a Large Shared Memory Production Workload," presented at the Revised Papers from the 7th International Workshop on Job Scheduling Strategies for Parallel Processing, 2001.

[142] K. Windisch, V. Lo, D. Feitelson, R. Moore, and B. Nitzberg, "A Comparison of Workload Traces from Two Production Parallel Machines," presented at the Proceedings of the 6th Symposium on the Frontiers of Massively Parallel Computation, 1996.

[143] U. Lublin and D. G. Feitelson, "The workload on parallel supercomputers: modeling the characteristics of rigid jobs," *Journal of Parallel and Distributed Computing*, vol. 63, pp. 1105-1122, 2003.

[144] J. Jann, P. Pattnaik, H. Franke, F. Wang, J. Skovira, and J. Riordan, "Modeling of Workload in MPPs," presented at the Proceedings of the Job Scheduling Strategies for Parallel Processing, 1997.

[145] W. Cirne and F. Berman, "A comprehensive model of the supercomputer workload," in *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*, 2001, pp. 140-148.

[146] L. M. P. F. a. H. Rudov, "Model of grid scheduling problem," in *In AAAI Press Technical Reports, editor, In AAAI05 Workshop on Exploring Planning and Scheduling for Web Services, Grid and Autonomic Computing*, 2005.

[147] T. G. L. H. El-Rewini, and H. H. Ali, *Task scheduling in parallel and distributed systems*: Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1994.

[148] J. Blazewicz, Ecker, K.H., Pesch, E., Schmidt, G. und J. Weglarz, *Scheduling Computer and Manufacturing Processes*: Berlin (Springer), 2001.

[149] I. Rodero, F. Guim, and J. Corbalan, "Evaluation of Coordinated Grid Scheduling Strategies," in *High Performance Computing and Communications, 2009. HPCC '09. 11th IEEE International Conference on*, 2009, pp. 1-10.

[150] M. A. Bender, S. Chakrabarti, and S. Muthukrishnan, "Flow and stretch metrics for scheduling continuous job streams," presented at the Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms, San Francisco, California, United States, 1998.

[151] A. Shafi, B. Carpenter, and M. Baker, "Nested parallelism for multi-core HPC systems using Java," *Journal of Parallel and Distributed Computing*, vol. 69, pp. 532-545, 2009.

[152] C.-L. Li and X. Wang, "Scheduling parallel machines with inclusive processing set restrictions and job release times," *European Journal of Operational Research*, vol. 200, pp. 702-710, 2010.

[153] A. Attanasio, G. Ghiani, L. Grandinetti, E. Guerriero, and F. Guerriero, "Operations research methods for resource management and scheduling in a computational grid: a survey," in *Advances in Parallel Computing*. vol. Volume 14, G. Lucio, Ed., ed: North-Holland, 2005, pp. 53-81.

[154] S. J. Chapin, W. Cirne, D. G. Feitelson, J. P. Jones, S. T. Leutenegger, U. Schwiegelshohn, W. Smith, and D. Talby, "Benchmarks and Standards for the Evaluation of Parallel Job Schedulers," presented at the Proceedings of the Job Scheduling Strategies for Parallel Processing, 1999.

[155] D. Feitelson. *Parallel Workloads Archive*. Available: http://www.cs.huji.ac.il/labs/parallel/workload/ [Apr. 07, 2009].

[156] JTransforms. Available: http://sites.google.com/site/piotrwendykier/software/jtransforms [Feb. 05, 2010].

[157] JChart2D. Available: http://jchart2d.sourceforge.net/index.shtml

[158] Commons-Math. *Commons-Math: The Apache Commons Mathematics Library*. Available: http://commons.apache.org/math/index.html [Feb. 05, 2010].

[159] H. Li, "Workload dynamics on clusters and grids," *The Journal of Supercomputing*, vol. 47, pp. 1-20, 2009.

[160] G. L. p. s. a. n. t. J. o. G. I. M. Gable, 6, 3–4.

[161] *Worldwide LHC Computing Grid*. Available: http://lcg.web.cern.ch/lcg/ [Apr. 07, 2009].

[162] *AuverGrid*. Available: http://www.auvergrid.fr/ [Apr. 07, 2009].

[163] P. Dumon. (1998). *The Perfect OS*. Available: http://tunes.org/unios/stdsched.html [May. 09, 2010].

LIST OF PUBLICATIONS

[1] S. N. M. Shah, A. K. B. Mahmood, A. Oxley, "Robust Grid Scheduling," in *2009 National Post Graduate Conference, NPC 2009*.

[2] S. N. M. Shah, A. K. B. Mahmood, A. Oxley, "Hybrid Scheduling and Dual Queue Scheduling," in *Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference on*, 2009, pp. 539-543.

[3] S. N. M. Shah, A. K. B. Mahmood, A. Oxley, "Hybrid Resource Allocation Method for Grid Computing," 2010 The 2nd International Conference on Computer Research and Development (ICCRD 2010), IEEE, 7-9 May 2010, Kuala Lumpur, Malaysia, pp. 426-431, 2010.

[4] S. N. M. Shah, A. K. B. Mahmood, A. Oxley, "Analysis and evaluation of Grid scheduling algorithms using real workload traces," The International ACM Conference on Management of Emergent Digital EcoSystems(ACM MEDES'10), 26-29 October 2010, Bangkok, Thailand.

[5] S. N. M. Shah, A. K. B. Mahmood, A. Oxley, "Modified Least Cost Method for Grid Resource Allocation," International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery(CyberC 2010), IEEE, 10-11 October 2010, HuangShan, AnHui, China, pp. 218-225, 2010.

[6] S. N. M. Shah, A. K. B. Mahmood, A. Oxley, "Development and Performance Analysis of Grid Scheduling Algorithms," in *Advances in Information Technology*. vol. 55, B. Papasratorn, *et al.*, Eds., ed: Springer Berlin Heidelberg, 2009, pp. 170-181.

[7] S. N. M. Shah, A. K. B. Mahmood, A. Oxley, "Dynamic Multilevel Hybrid Scheduling Algorithms for Grid Computing," *Procedia Computer Science, Elsevier*, vol. 4, pp. 402-411, 2011.

[8] S. N. M. Shah, A. K. B. Mahmood, A. Oxley,"Dynamic Multilevel Dual Queue Scheduling Algorithms for Grid Computing," in *Software Engineering and Computer Systems*. vol. 179, J. Mohamad Zain, *et al.*, Eds., ed: Springer Berlin Heidelberg, 2011, pp. 425-440.

[9] S. N. M. Shah, A. K. B. Mahmood, A. Oxley,"SYEDWSIM: A Web Based Simulator for Grid Workload Analysis," in *Software Engineering and Computer Systems*. vol. 181, J. M. Zain, *et al.*, Eds., ed: Springer Berlin Heidelberg, 2011, pp. 677-692.

[10] S. N. M. Shah, A. K. B. Mahmood, A. Oxley, "Development and performance analysis of Grid resource allocation methods," *Int. J. Information Technology, Communications and Convergence, InderScience,* 2011

APPENDIX A

Grid Scheduling Algorithms

A.1 Proposed Resource Allocation Method

To describe the algorithm, the notations (Z_{min}, *Smallest*, A_p, B_q, T_{pq}, $T_{pq} \xrightarrow[mapping]{} P_q$) used in the Modified Least Cost Method are described below:

- Z_{min}: It is the total allocation cost.
- *Smallest*: It is variable which keeps the smallest cost cell(s) from the resource allocation table. Min function is applied to find out the *smallest* while ignoring any row where the unallocated workload is zero ($A_i = 0$), and any column where the unallocated processor capacity is zero ($B_j = 0$)
- A_p: remaining or total workload demand of job *p* corresponding to *cell(p, q)*
- B_q: available or total processing capability of processor *q* corresponding to *cell(p, q)*
- T_{pq}: It is estimated task length of the job workload (i.e. A_p) that can be allocated to a processor P_q holding processor availability B_q
- $T_{pq} \xrightarrow[mapping]{} P_q$: Mapping the task T_{pq} of job J_p to processor P_q
- $A_i = 0$: If all of the A_i are zero then stop as this means that a solution has been found, otherwise continue further allocations

A.1.1 Procedure MLCM

Begin

Step1 initialization

$Z_{min} := 0$

Repeat

Step2 Find the least cost cell

$Smallest := \min\{C_{ij} : A_i \neq 0, B_j \neq 0\} \ \forall \ i = 1,2,...m \ , \ j = 1,2,..n$

$count = size\{C_{ij} : C_{ij} = smallest\}$

If (count>1) Then

select the *cell* which does not include the *next smallest* in its corresponding row or column

If a tie occurs for this case Then

select the *cell* which can host the minimum *job workload*

End If

End If

Step3 mapping of task to processor

$$T_{pq} := \begin{cases} A_p \ If \ A_p \leq B_q \\ B_q \ If \ A_p > B_q \end{cases}$$

$T_{pq} \xrightarrow[mapping]{} P_q$

$A_p := A_p - T_{pq}$

$B_q := B_q - T_{pq}$

$Z_{\min} := Z_{\min} + T_{pq} \ X \ Smallest$

Until $A_i = 0 \ \ \forall i$

End

A.2 Proposed Job Scheduling Algorithms

Let the notations used in the job scheduling algorithms are (P, n_1, n_2, t_{exec} , t_{large} , w, k, E, Q, W, T, Q).

- P: set of processes in the system. Formally it can be written as:

$$P = \{P_1, P_2, P_3 \ldots\ldots P_n\} \quad \forall P_i \subseteq P \wedge i \in N$$

where 'N' is set of natural number and 'n' denotes number of processes in the ready queue.

  - Each process Pi is described by its Process Control Block (PCB) which includes process id, arrival time, CPU burst time, execution time and state. Formally it can be represented as:

$$P_i < id, arrivaltime, cputime, executiontime, state >$$

where

- '*id*' represents the Process id
- '*arrivaltime*' denotes the time at which process Pi is arrived in the system
- '*cputime*' shows the demand by the process Pi to execute at CPU level
- '*executiontime*' shows time units for which process Pi is already executed at CPU
- '*state*' shows the current state of a process Pi.

  - A process can be in one of the following states:

$$state \in \{0, 1, 2, 3, 4, 5\}$$

where

- *'0'* shows that process is newly created
- *'1'* represents that process is in ready queue (or execution queue) and not executed yet

- *'2'* shows that process is in ready queue (or execution queue) and partially executed
- *'3'* shows that process is successfully executed
- *'4'* denotes that process is failed during execution
- *'5'* represents the process in the waiting queue (for dual queue scheduling algorithms)

o For example; if $P_i \in < 131, 201, 79, 25, 2 >$ then

It means process P_i possesses the following attribute values. Process id is *'131'*; process arrived at *'201'* clock interval, *'79'* units of CPU time demanded for execution, P_i is already executed for '*25*' units of CPU time and process in the ready queue and partially executed. Process P_i needs *'54'* units more for execution at CPU.

- n_1 : number of processes in the waiting queue
- n_2 : number of processes in the execution queue
- t_{exec} : sum of execution time of processes
- t_{large} : cpu burst time of longest process in the ready queue
- *w*: logical counter to manage execution of longest process. if *w=1*; it means a process with longest CPU time will be given a turn for execution; and counter t_{exec} is reset to zero
- *k*: counter variable used to manage the flow of execution of sorted current processes in the ready queue
- *E*: set of processes, which are executed successfully by the algorithm. E is also called safe sequence of processes.
- *Q*: set of processes in the execution queue
- *W*: set of processes in the waiting queue
- *TQ*: denotes the time quantum. It is a user defined fixed value for MH and MDQ scheduling algorithms; whilst TQ is dynamic value for MHM, MHR, MDQM and MDQR.

o Dynamic Time Quantum will be computed by taking median of CPU burst times values of present processes in the ready queue for MHM and MDQM:

$$TQ = median(P_1.cputime, P_2.cputime, P_3.cputime, ...P_n.cputime)$$

- Dynamic Time Quantum will be computed by taking square root on average of CPU burst times values of processes that are present in the ready queue for MHR and MDQR:

$$TQ = sqrt(avg(P_1.cputime, P_2.cputime, P_3.cputime, ...P_n.cputime))$$

A.2.1 Resource Allocation and Job Distribution Strategy

Procedure *Master_Process*

Begin

1. Master process reads the workload trace file and stores into a dynamic array.
2. Using dynamic looping strategy, it distributes the part of the workload dynamically depending upon the total number of available processors (i.e., computational nodes) for the computation.
3. It sends the distributed workload data to slave processors using round robin algorithm. It maintains the counter for each processor to keep the information about the number of jobs allocated to each slave processor.
4. Then, master process waits till the computation of performance parameters of the scheduling algorithm (also known as slave process) at each slave processor is over. Slave processors will send back the parameter values back to the master.
5. It receives the performance parameters (i.e., results) computed by the employed job scheduling algorithm at each slave processor (also called worker).

   MPI.COMM_WORLD.Recv(result, 0, 9, MPI.DOUBLE, worker, 99)
6. After receiving, it computes the summation of the values for the performance parameters of waiting times, turnaround time, response times and slowdown times, taken from each slave processor.
7. It also computes the maximum values, out of all the maximal values for the total completion times as well as for the job stretch times, taken from each slave processor.

8. It calculates the average values for performance factors of waiting times, turnaround time, response times and slowdown times.

End_Master_process

A.2.2 Procedure Multilevel Hybrid Scheduling and Dynamic Multilevel Hybrid Scheduling

Procedure MH/MHM/MHR

Phase 1: ***Master Process (Allocation Strategy)***

Call Master_Process

Phase 2: ***Slave process (Job execution strategy)***

Begin

Step1 initialization

Let $P := \{\}$

Let $E := \{\}$

Let $t_{exec} := 0$

Let $t_{large} := 0$

Step2 Process arrivals

if new process Pi is arrived then

If $(P_i.state == 0)$ *then*

$set\ P_i.state = 1$

$P = P \bigcup P_i$

$n = n + 1$

endif

endif

Step 3 Sorting of jobs

$$Sorting\ \_\ A\lg orithm\ (P)_{CPU\ \_burstime\ .Asc} \xrightarrow{sort} P$$

Step 4 Time quantum strategy

// user defined time quantum for MDQ

$TQ = fixed\ \_value$

or

//TQ value for MHM

$TQ = median(P_1.cputime, P_2.cputime, ...P_n.cputime)$

or

// TQ value for MHR

$TQ = sqrt(avg(P_1.cputime, P_2.cputime, ...P_n.cputime))$

Step 5 execution Strategy

while(P! = NULL)

$set\ t_{large} = P_n.cputime$

$set\ w = 0$

$set\ k = 1$

$while(k \le n)$

Begin

$if\ (P_k.status == 1 \vee P_k.status == 2)then$

$if\ (P_k.cputime \le TQ)then$

$P_k.executiontime = P_k.executiontime + P_k.cputime$

$P_k.state = 3$

$E = E \bigcup P_k$

$P = P - P_k$

$n = n - 1$

$t_{exec} = t_{exec} + P_k.executiontime$

else

$P_k.executiontime = P_k.executiontime + TQ$

$P_k.state = 2$

$t_{exec} = t_{exec} + P_k.executiontime$

endif

endif

$If(w == 1)$

$t_{exec} = 0$

$w = 0$

$break - while - loop$

$goto - step - 3$

endif

$if\ (t_{exec} \geq t_{l\arg e})then$

$k = n$

$w = 1$

else

if (New processes arrived in P) then

$break - while - loop$

$goto - step - 2$

else

$k = k + 1$

endif

endif

$wend\ (inner\ \_\ loop)$

$end\ \_\ main\ \_\ while\ \_\ loop$

Step 6 Compute performance parameters

Compute performance parameters - waiting times, turnaround times, response times, slowdown times, total completion times and maximum job stretch times

Step 7 Send results back to the master process

Send computed performance parameters (i.e., output) back to master processor whose processor id is '0'.

MPI.COMM_WORLD.Isend(output, 0, 9, MPI.DOUBLE, 0, 99)

End_Slave_process

A.2.3 Procedure Multilevel Dual Queue Scheduling and Dynamic Multilevel Dual Queue Scheduling

Procedure MDQ/ MDQM/MDQR

Phase 1: Master Process (Allocation Strategy)

Call Master_Process

Phase 2: Slave process (Job execution strategy)

Begin

Step 1 initialization

Let $W := \{\}$

Let $Q := \{\}$

Let $E := \{\}$

Let $t_{exec} := 0$

Let $t_{large} := 0$

Step 2 Process arrivals in the waiting queue

if new process P_i is arrived then

If $(P_i.state == 0)$*then*

$set\ P_i.state = 5$

$W = W \bigcup P_i$

$n_1 = n_1 + 1$

endif

endif

Step 3 Process arrivals in the execution queue

if (execution queue is empty) or (longest process is given the turn for execution) then

while(W! = NULL)

If $(P_i.state == 5)$*then*

$set\ P_i.state = 1$

$W = W - P_i$

$P = P \bigcup P_i$

$n_1 = n_1 - 1$

$n_2 = n_2 + 1$

endif

end_while_loop

endif

Step 4 Sorting of processes in the execution queue

$Sorting\_Algorithm\_(Q)_{CPU\_burstime.Asc} \xrightarrow{sort} Q$

Step 5 Time quantum calculation

// user defined time quantum for MDQ

TQ = fixed_value

or

//time quantum for MDQM

$TQ = median(P_1.cputime, P_2.cputime, ...P_n.cputime)$

or

//time quantum for MDQR

$TQ = sqrt(avg(P_1.cputime, P_2.cputime, ...P_n.cputime))$

Step 6 execution strategy

while(Q! = NULL)

$set\ t_{large} = P_n.cputime$

$set\ w = 0$

$set\ k = 1$

$while(k \le n)$

Begin

$if\ (P_k.status == 1 \vee P_k.status == 2)then$

$if\ (P_k.cputime \le TQ)then$

$P_k.executiontime = P_k.executiontime + P_k.cputime$

$P_k.state = 3$

$E = E \bigcup P_k$

$Q = Q - P_k$

$n_2 = n_2 - 1$

$t_{exec} = t_{exec} + P_k.executiontime$

else

$P_k.executiontime = P_k.executiontime + TQ$

$P_k.state = 2$

$t_{exec} = t_{exec} + P_k.executiontime$

endif

endif

$If(w==1)$

$t_{exec} = 0$

$w = 0$

$break–while–loop$

$goto–step2$

$endif$

$if\ (t_{exec} \geq t_{l\arg e})then$

$k = n$

$w = 1$

$else$

if *(New processes arrived in W) then*

$break–while–loop$

$goto - step\ 2$

$else$

$k = k + 1$

$endif$

$endif$

$wend(inner\_loop)$

$end\_main\_while\_loop$

Step 7 Compute performance parameters

Compute performance parameters - waiting times, turnaround times, response times, slowdown times, total completion times and maximum job stretch times

Step 8 Send results back to the master process

Send computed performance parameters (output) back to master processor (processor id is '0').

MPI.COMM_WORLD.Isend(output, 0, 9, MPI.DOUBLE, 0, 99);

End_slave_process

APPENDIX B

Format of Real Grid Workloads

B.1 Format of LCG1 workload

```
# Log source:  Parallel Workloads Archive
#
#-------------------------------------------------------------
# Format documentation: Grid Workload Format (http://gwa.ewi.tudelft.nl/)
# Field description from left to right:
#
# 1  JobID              counter
# 2  SubmitTime         in seconds, starting from zero
# 3  WaitTime           in seconds
# 4  RunTime            runtime measured in wallclock seconds
# 5  NProcs             number of allocated processors
# 6  AverageCPUTimeUsed   average of CPU time over all allocated processors
# 7  Used Memory        average per processor in kilobytes
# 8  ReqNProcs          requested number of processors
# 9  ReqTime:           requested time measured in wallclock seconds
# 10 ReqMemory          requested memory (average per processor)
# 11 Status             job completed = 1, job failed = 0, job cancelled = 5
# 12 UserID             string identifier for user
# 13 GroupID            string identifier for group user belongs to
# 14 ExecutableID       name of executable
# 15 QueueID            string identifier for queue
# 16 PartitionID        string identifier for partition
# 17 OrigSiteID         string identifier for submission site
```

18 LastRunSiteID string identifier for execution site

19 JobStructure single job = UNITARY, composite job = BoT

20 JobStructureParams if JobStructure = BoT, contains batch identifier

21 UsedNetwork used network resources in kilobytes/second

22 UsedLocalDiskSpace in megabytes

23 UsedResources list of comma-separated generic resources (ResourceDescription:Consumption)

c.q. memory usage in Gb seconds, io data transferred, and io wait time in seconds

24 ReqPlatform CPUArchitecture,OS,OSVersion

25 ReqNetwork in kilobytes/second

26 ReqLocalDiskSpace in megabytes

27 ReqResources list of comma-separated generic resources (ResourceDescription:Consumption)

28 VOID identifier for Virtual Organization

29 ProjectID identifier for project

#

(fields contain -1 if not available)

1 1132444805 -1 83 1 -1 -1 -1 -1 -1 -1
U1 G1 -1 -1 1 SWF SWF -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1

2 1132444808 -1 3611 1 -1 -1 -1 -1 -1 -1
U2 G2 -1 -1 2 SWF SWF -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1

3 1132444817 -1 205 1 -1 -1 -1 -1 -1 -1
U1 G1 -1 -1 3 SWF SWF -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1

4 1132444819 -1 130 1 -1 -1 -1 -1 -1 -1
U2 G2 -1 -1 2 SWF SWF -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1

5 1132444825 -1 969 1 -1 -1 -1 -1 -1 -1
U3 G3 -1 -1 3 SWF SWF -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1

6	1132444829	-1	129	1	-1	-1	-1	-1	-1	-1	
	U2	G2	-1	-1	2	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
7	1132444830	-1	201	1	-1	-1	-1	-1	-1	-1	
	U1	G1	-1	-1	4	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
8	1132444839	-1	10707	1	-1	-1	-1	-1	-1	-1	
	U2	G2	-1	-1	2	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
9	1132444842	-1	79	1	-1	-1	-1	-1	-1	-1	
	U1	G1	-1	-1	5	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
10	1132444843	-1	1908	1	-1	-1	-1	-1	-1	-1	
	U3	G3	-1	-1	3	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
11	1132444850	-1	9885	1	-1	-1	-1	-1	-1	-1	
	U2	G2	-1	-1	2	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
12	1132444852	-1	78	1	-1	-1	-1	-1	-1	-1	
	U4	G4	-1	-1	6	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
13	1132444857	-1	74	1	-1	-1	-1	-1	-1	-1	
	U1	G1	-1	-1	7	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
14	1132444859	-1	10006	1	-1	-1	-1	-1	-1	-1	
	U2	G2	-1	-1	2	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
15	1132444859	-1	22	1	-1	-1	-1	-1	-1	-1	
	U4	G4	-1	-1	8	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
16	1132444864	-1	1972	1	-1	-1	-1	-1	-1	-1	
	U3	G3	-1	-1	3	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				

17	1132444866	-1	73	1	-1	-1	-1	-1	-1	-1	
	U4	G4	-1	-1	9	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
18	1132444867	-1	142	1	-1	-1	-1	-1	-1	-1	
	U1	G1	-1	-1	10	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
19	1132444872	-1	5050	1	-1	-1	-1	-1	-1	-1	
	U2	G2	-1	-1	2	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
20	1132444874	-1	311	1	-1	-1	-1	-1	-1	-1	
	U4	G4	-1	-1	11	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
21	1132444881	-1	78	1	-1	-1	-1	-1	-1	-1	
	U4	G4	-1	-1	12	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
22	1132444882	-1	9951	1	-1	-1	-1	-1	-1	-1	
	U2	G2	-1	-1	2	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
23	1132444883	-1	198	1	-1	-1	-1	-1	-1	-1	
	U1	G1	-1	-1	13	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
24	1132444888	-1	77	1	-1	-1	-1	-1	-1	-1	
	U4	G4	-1	-1	14	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
25	1132444891	-1	11010	1	-1	-1	-1	-1	-1	-1	
	U2	G2	-1	-1	2	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
26	1132444892	-1	209	1	-1	-1	-1	-1	-1	-1	
	U1	G1	-1	-1	15	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
27	1132444892	-1	1483	1	-1	-1	-1	-1	-1	-1	
	U3	G3	-1	-1	3	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				

28	1132444895	-1	199	1	-1	-1	-1	-1	-1	-1	
	U4	G4	-1	-1	16	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
29	1132444901	-1	9652	1	-1	-1	-1	-1	-1	-1	
	U2	G2	-1	-1	2	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
30	1132444902	-1	78	1	-1	-1	-1	-1	-1	-1	
	U4	G4	-1	-1	17	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
31	1132444907	-1	134	1	-1	-1	-1	-1	-1	-1	
	U1	G1	-1	-1	18	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
32	1132444909	-1	856	1	-1	-1	-1	-1	-1	-1	
	U3	G3	-1	-1	3	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
33	1132444912	-1	9887	1	-1	-1	-1	-1	-1	-1	
	U2	G2	-1	-1	2	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
34	1132444914	-1	136	1	-1	-1	-1	-1	-1	-1	
	U1	G1	-1	-1	19	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
35	1132444922	-1	205	1	-1	-1	-1	-1	-1	-1	
	U2	G2	-1	-1	2	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
36	1132444929	-1	2028	1	-1	-1	-1	-1	-1	-1	
	U3	G3	-1	-1	3	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
37	1132444930	-1	67	1	-1	-1	-1	-1	-1	-1	
	U1	G1	-1	-1	18	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
38	1132444932	-1	179	1	-1	-1	-1	-1	-1	-1	
	U2	G2	-1	-1	2	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				

39	1132444936	-1	195	1	-1	-1	-1	-1	-1	-1	
	U1	G1	-1	-1	4	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
40	1132444942	-1	9886	1	-1	-1	-1	-1	-1	-1	
	U2	G2	-1	-1	2	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
41	1132444952	-1	1032	1	-1	-1	-1	-1	-1	-1	
	U3	G3	-1	-1	3	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
42	1132444953	-1	10192	1	-1	-1	-1	-1	-1	-1	
	U2	G2	-1	-1	2	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
43	1132444956	-1	144	1	-1	-1	-1	-1	-1	-1	
	U1	G1	-1	-1	13	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
44	1132444958	-1	135	1	-1	-1	-1	-1	-1	-1	
	U1	G1	-1	-1	10	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
45	1132444964	-1	9651	1	-1	-1	-1	-1	-1	-1	
	U2	G2	-1	-1	2	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
46	1132444970	-1	139	1	-1	-1	-1	-1	-1	-1	
	U3	G3	-1	-1	3	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
47	1132444974	-1	179	1	-1	-1	-1	-1	-1	-1	
	U2	G2	-1	-1	2	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
48	1132444979	-1	139	1	-1	-1	-1	-1	-1	-1	
	U1	G1	-1	-1	20	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
49	1132444983	-1	5116	1	-1	-1	-1	-1	-1	-1	
	U2	G2	-1	-1	2	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				

50	1132444993	-1	9711	1	-1	-1	-1	-1	-1	-1	
	U2	G2	-1	-1	2	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
51	1132444993	-1	974	1	-1	-1	-1	-1	-1	-1	
	U3	G3	-1	-1	3	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
52	1132445002	-1	18	1	-1	-1	-1	-1	-1	-1	
	U1	G1	-1	-1	21	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
53	1132445002	-1	3437	1	-1	-1	-1	-1	-1	-1	
	U2	G2	-1	-1	2	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
54	1132445003	-1	84	1	-1	-1	-1	-1	-1	-1	
	U1	G1	-1	-1	22	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
55	1132445012	-1	10590	1	-1	-1	-1	-1	-1	-1	
	U2	G2	-1	-1	2	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
56	1132445016	-1	974	1	-1	-1	-1	-1	-1	-1	
	U3	G3	-1	-1	3	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
57	1132445023	-1	192	1	-1	-1	-1	-1	-1	-1	
	U2	G2	-1	-1	2	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
58	1132445024	-1	136	1	-1	-1	-1	-1	-1	-1	
	U1	G1	-1	-1	23	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
59	1132445025	-1	138	1	-1	-1	-1	-1	-1	-1	
	U1	G1	-1	-1	23	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
60	1132445033	-1	194	1	-1	-1	-1	-1	-1	-1	
	U2	G2	-1	-1	2	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				

61	1132445034	-1	2033	1	-1	-1	-1	-1	-1	-1	
	U3	G3	-1	-1	3	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
62	1132445042	-1	7185	1	-1	-1	-1	-1	-1	-1	
	U2	G2	-1	-1	2	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
63	1132445047	-1	137	1	-1	-1	-1	-1	-1	-1	
	U1	G1	-1	-1	10	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
64	1132445047	-1	80	1	-1	-1	-1	-1	-1	-1	
	U1	G1	-1	-1	24	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
65	1132445051	-1	1606	1	-1	-1	-1	-1	-1	-1	
	U3	G3	-1	-1	3	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
66	1132445052	-1	9579	1	-1	-1	-1	-1	-1	-1	
	U2	G2	-1	-1	2	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
67	1132445062	-1	198	1	-1	-1	-1	-1	-1	-1	
	U2	G2	-1	-1	2	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
68	1132445068	-1	906	1	-1	-1	-1	-1	-1	-1	
	U3	G3	-1	-1	3	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
69	1132445071	-1	139	1	-1	-1	-1	-1	-1	-1	
	U1	G1	-1	-1	25	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
70	1132445071	-1	74	1	-1	-1	-1	-1	-1	-1	
	U1	G1	-1	-1	23	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
71	1132445073	-1	9100	1	-1	-1	-1	-1	-1	-1	
	U2	G2	-1	-1	2	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				

72	1132445083	-1	10784	1	-1	-1	-1	-1	-1	-1
	U2	G2	-1	-1	2	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
73	1132445086	-1	849	1	-1	-1	-1	-1	-1	-1
	U3	G3	-1	-1	3	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
74	1132445093	-1	10374	1	-1	-1	-1	-1	-1	-1
	U2	G2	-1	-1	2	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
75	1132445093	-1	83	1	-1	-1	-1	-1	-1	-1
	U1	G1	-1	-1	26	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
76	1132445095	-1	139	1	-1	-1	-1	-1	-1	-1
	U1	G1	-1	-1	19	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
77	1132445102	-1	9830	1	-1	-1	-1	-1	-1	-1
	U2	G2	-1	-1	2	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
78	1132445103	-1	1663	1	-1	-1	-1	-1	-1	-1
	U3	G3	-1	-1	3	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
79	1132445111	-1	9817	1	-1	-1	-1	-1	-1	-1
	U2	G2	-1	-1	2	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
80	1132445115	-1	80	1	-1	-1	-1	-1	-1	-1
	U1	G1	-1	-1	27	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
81	1132445118	-1	137	1	-1	-1	-1	-1	-1	-1
	U1	G1	-1	-1	28	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
82	1132445120	-1	916	1	-1	-1	-1	-1	-1	-1
	U3	G3	-1	-1	3	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				

83	1132445121	-1	138	1	-1	-1	-1	-1	-1	-1	
	U2	G2	-1	-1	2	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
84	1132445134	-1	128	1	-1	-1	-1	-1	-1	-1	
	U2	G2	-1	-1	29	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
85	1132445138	-1	75	1	-1	-1	-1	-1	-1	-1	
	U1	G1	-1	-1	30	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
86	1132445143	-1	1739	1	-1	-1	-1	-1	-1	-1	
	U3	G3	-1	-1	3	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
87	1132445144	-1	143	1	-1	-1	-1	-1	-1	-1	
	U2	G2	-1	-1	29	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
88	1132445154	-1	10883	1	-1	-1	-1	-1	-1	-1	
	U2	G2	-1	-1	29	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
89	1132445160	-1	133	1	-1	-1	-1	-1	-1	-1	
	U1	G1	-1	-1	31	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
90	1132445162	-1	1192	1	-1	-1	-1	-1	-1	-1	
	U3	G3	-1	-1	3	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
91	1132445164	-1	139	1	-1	-1	-1	-1	-1	-1	
	U2	G2	-1	-1	29	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
92	1132445173	-1	13643	1	-1	-1	-1	-1	-1	-1	
	U2	G2	-1	-1	29	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
93	1132445182	-1	134	1	-1	-1	-1	-1	-1	-1	
	U1	G1	-1	-1	32	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				

94	1132445182		-1	13698	1	-1	-1	-1	-1	-1	-1
	U2	G2	-1	-1	29	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
95	1132445182		-1	1420	1	-1	-1	-1	-1	-1	-1
	U3	G3	-1	-1	3	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
96	1132445191		-1	12939	1	-1	-1	-1	-1	-1	-1
	U2	G2	-1	-1	29	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
97	1132445201		-1	20324	1	-1	-1	-1	-1	-1	-1
	U2	G2	-1	-1	29	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
98	1132445205		-1	200	1	-1	-1	-1	-1	-1	-1
	U1	G1	-1	-1	15	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
99	1132445207		-1	1374	1	-1	-1	-1	-1	-1	-1
	U3	G3	-1	-1	3	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				
100	1132445211		-1	6940	1	-1	-1	-1	-1	-1	-1
	U2	G2	-1	-1	29	SWF	SWF	-1	-1	-1	-1
	-1	-1	-1	-1	-1	-1	-1				

B.2 Format of AuverGrid workload

System name: LPC
System info: Laboratoire de Physique Corpusculaire - Part of the LCG (Large hadron collider Computing Grid project)
Sites: 5
Processors: 475
CPU Info: 3 GHZ Pentium IV Xeon Linux Cluster
Memory: ?
Disk space: ?
Network: ?
Log source: Local resource manager
#
#---
Format documentation: Grid Workload Format (http://gwa.ewi.tudelft.nl/)
Field description from left to right:
#
1 JobID counter
2 SubmitTime in seconds, starting from zero
3 WaitTime in seconds
4 RunTime runtime measured in wallclock seconds
5 NProcs number of allocated processors
6 AverageCPUTimeUsed average of CPU time over all allocated processors
7 Used Memory average per processor in kilobytes
8 ReqNProcs requested number of processors
9 ReqTime: requested time measured in wallclock seconds
10 ReqMemory requested memory (average per processor)
11 Status job completed = 1, job failed = 0, job cancelled = 5
12 UserID string identifier for user
13 GroupID string identifier for group user belongs to
14 ExecutableID name of executable
15 QueueID string identifier for queue
16 PartitionID string identifier for partition

17 OrigSiteID string identifier for submission site
18 LastRunSiteID string identifier for execution site
19 JobStructure single job = UNITARY, composite job = BoT
20 JobStructureParams if JobStructure = BoT, contains batch identifier
21 UsedNetwork used network resources in kilobytes/second
22 UsedLocalDiskSpace in megabytes
23 UsedResources list of comma-separated generic resources
(ResourceDescription:Consumption)
c.q. memory usage in Gb seconds, io data transferred, and io wait time in seconds
24 ReqPlatform CPUArchitecture,OS,OSVersion
25 ReqNetwork in kilobytes/second
26 ReqLocalDiskSpace in megabytes
27 ReqResources list of comma-separated generic resources
(ResourceDescription:Consumption)
28 VOID identifier for Virtual Organization
29 ProjectID identifier for project
#
(fields contain -1 if not available)
#
#---
#
1 1136070024 203761 138467 1 138371 98652 1
259200 -1 1 U2004S1 G3 X1 Q5 1
clrlcgce02 clrlcgce02 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1
2 1136070690 0 11 1 4 35848 1 259200 -1
1 U1023S0 G1 X1 Q1 1 clrlcgce01 clrlcgce01
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
3 1136071207 117 201203 1 0 0 1 259200
-1 1 U2035S1 G6 X1 Q2 1 clrlcgce02
clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1

4 1136071267 4406 196985 1 0 0 1 259200
-1 1 U2035S1 G6 X1 Q2 1 clrlcgce02
clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1
5 1136071269 202516 19520 1 18731 522268 1
259200 -1 1 U2035S1 G6 X1 Q2 1
clrlcgce02 clrlcgce02 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1
6 1136072890 1 46 1 3 49216 1 900 -1 1
U1018S0 G1 X1 Q2 1 clrlcgce01 clrlcgce01 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1
7 1136074263 2629516 21 1 0 35712 1 5400 -1
1 U5005S3 G1 X1 Q5 1 iut15 iut15 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1
8 1136074695 1 197831 1 0 0 1 259200
-1 1 U1033S0 G6 X1 Q4 1 clrlcgce01
clrlcgce01 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1
9 1136074754 2 197682 1 0 0 1 259200
-1 1 U1033S0 G6 X1 Q4 1 clrlcgce01
clrlcgce01 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1
10 1136074756 2 197552 1 0 0 1 259200
-1 1 U1033S0 G6 X1 Q4 1 clrlcgce01
clrlcgce01 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1
11 1136074814 1 197495 1 0 0 1 259200
-1 1 U1033S0 G6 X1 Q4 1 clrlcgce01
clrlcgce01 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1
12 1136076162 197623 18799 1 18198 480484 1
259200 -1 1 U2035S1 G6 X1 Q2 1

clrlcgce02 clrlcgce02 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1
13 1136076694 2627085 21 1 0 35720 1 5400 -1
1 U5005S3 G1 X1 Q2 1 iut15 iut15 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1
14 1136077528 58 -1 0 -1 -1 1 -1 -1 5
-1 -1 -1 Q3 1 clrlcgce01 clrlcgce01 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1
15 1136077551 975 193984 1 0 0 1 259200
-1 1 U2037S1 G4 X1 Q1 1 clrlcgce02
clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1
16 1136077775 58 -1 0 -1 -1 1 -1 -1 5
-1 -1 -1 Q2 1 clrlcgce01 clrlcgce01 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1
17 1136077783 196002 2 1 0 0 1 900 -1
1 U2031S1 G1 X1 Q3 1 clrlcgce02 clrlcgce02
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
18 1136077889 58 -1 0 -1 -1 1 -1 -1 5
-1 -1 -1 Q3 1 clrlcgce01 clrlcgce01 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1
19 1136078195 58 -1 0 -1 -1 1 -1 -1 5
-1 -1 -1 Q2 1 clrlcgce01 clrlcgce01 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1
20 1136078435 58 -1 0 -1 -1 1 -1 -1 5
-1 -1 -1 Q2 1 clrlcgce01 clrlcgce01 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1
21 1136078608 58 -1 0 -1 -1 1 -1 -1 5
-1 -1 -1 Q3 1 clrlcgce01 clrlcgce01 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1
22 1136078676 117 -1 0 -1 -1 1 -1 -1 5
-1 -1 -1 Q2 1 clrlcgce01 clrlcgce01 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1

23 1136078802 194983 2 1 0 6376 1 5400 -1
1 U2023S1 G1 X1 Q4 1 clrlcgce02 clrlcgce02
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
24 1136079148 58 -1 0 -1 -1 1 -1 -1 5
-1 -1 -1 Q3 1 clrlcgce01 clrlcgce01 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1
25 1136203888 2499978 20 1 0 35752 1 5400 -1
1 U5005S3 G1 X1 Q2 1 iut15 iut15 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1
26 1136204188 2499722 20 1 0 0 1 172800
-1 1 U5005S3 G1 X1 Q6 1 iut15 iut15 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1
27 1136212017 2491849 20 1 0 0 1 5400 -1
1 U5005S3 G1 X1 Q5 1 iut15 iut15 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1
28 1136229063 2474476 21 1 0 0 1 900 -1
1 U5005S3 G1 X1 Q3 1 iut15 iut15 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1
29 1136237461 2466405 20 1 0 35760 1 5400 -1
1 U5005S3 G1 X1 Q2 1 iut15 iut15 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1
30 1136247692 27481 -1 0 -1 -1 1 -1 -1 5
-1 -1 -1 Q2 1 iut15 iut15 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1
31 1136250422 24752 -1 0 -1 -1 1 -1 -1 5
-1 -1 -1 Q1 1 iut15 iut15 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1
32 1136260866 14307 -1 0 -1 -1 1 -1 -1 5
-1 -1 -1 Q1 1 iut15 iut15 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1
33 1136262396 12777 -1 0 -1 -1 1 -1 -1 5
-1 -1 -1 Q1 1 iut15 iut15 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1

34 1136272204 2 922 1 20 74696 1 172800 -1
1 U1030S0 G1 X1 Q5 1 clrlcgce01 clrlcgce01
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
35 1136272205 1580 912 1 20 117784 1 172800
-1 1 U2031S1 G1 X1 Q6 1 clrlcgce02
clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1
36 1136274078 1 19143 1 18070 490916 1 259200
-1 1 U1033S0 G6 X1 Q4 1 clrlcgce01
clrlcgce01 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1
37 1136274080 2 19432 1 18657 470052 1 259200
-1 1 U1033S0 G6 X1 Q4 1 clrlcgce01
clrlcgce01 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1
38 1136274120 1 476 1 239 961104 1 31980
614400 5 U2003S1 G3 X1 Q5 1 clrlcgce02
clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1
39 1136274123 1 473 1 198 937320 1 31980
614400 5 U2003S1 G3 X1 Q5 1 clrlcgce02
clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1
40 1136274123 1 59 1 11 7740 1 19980 614400
1 U2003S1 G3 X1 Q5 1 clrlcgce02 clrlcgce02
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
41 1136274241 1 501 1 245 965184 1 31980
614400 5 U2003S1 G3 X1 Q5 1 clrlcgce02
clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1
42 1136274244 1 498 1 250 956288 1 31980
614400 5 U2003S1 G3 X1 Q5 1 clrlcgce02

clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

43 1136274600 1 595 1 215 958608 1 31980 614400 5 U2003S1 G3 X1 Q5 1 clrlcgce02 clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

44 1136274607 0 602 1 280 935536 1 31980 614400 5 U2003S1 G3 X1 Q5 1 clrlcgce02 clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

45 1136274607 0 602 1 222 984764 1 31980 614400 5 U2003S1 G3 X1 Q5 1 clrlcgce02 clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

46 1136274607 1 602 1 253 950132 1 31980 614400 5 U2003S1 G3 X1 Q5 1 clrlcgce02 clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

47 1136274609 1 599 1 206 951048 1 31980 614400 5 U2003S1 G3 X1 Q5 1 clrlcgce02 clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

48 1136274609 1 599 1 215 951844 1 31980 614400 5 U2003S1 G3 X1 Q5 1 clrlcgce02 clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

49 1136274609 1 599 1 201 943396 1 31980 614400 5 U2003S1 G3 X1 Q5 1 clrlcgce02 clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

50 1136274611 1 597 1 210 953368 1 31980 614400 5 U2003S1 G3 X1 Q5 1 clrlcgce02

clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

51 1136274611 1 461 1 162 938652 1 31980 614400 5 U2003S1 G3 X1 Q5 1 clrlcgce02 clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

52 1136274611 1 597 1 279 949680 1 31980 614400 5 U2003S1 G3 X1 Q5 1 clrlcgce02 clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

53 1136274841 1 641 1 240 951624 1 31980 614400 5 U2003S1 G3 X1 Q5 1 clrlcgce02 clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

54 1136274847 2 505 1 173 944424 1 31980 614400 5 U2003S1 G3 X1 Q5 1 clrlcgce02 clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

55 1136274848 1 575 1 175 951172 1 31980 614400 5 U2003S1 G3 X1 Q5 1 clrlcgce02 clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

56 1136274848 1 610 1 251 952884 1 31980 614400 5 U2003S1 G3 X1 Q5 1 clrlcgce02 clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

57 1136274849 1 946 1 314 959352 1 31980 614400 5 U2003S1 G3 X1 Q5 1 clrlcgce02 clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

58 1136274850 0 574 1 258 959928 1 31980 614400 5 U2003S1 G3 X1 Q5 1 clrlcgce02

clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1

59 1136274850 0 923 1 503 977304 1 31980
614400 5 U2003S1 G3 X1 Q5 1 clrlcgce02
clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1

60 1136274851 1 537 1 205 941008 1 31980
614400 5 U2003S1 G3 X1 Q5 1 clrlcgce02
clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1

61 1136274852 1 920 1 478 983184 1 31980
614400 5 U2003S1 G3 X1 Q5 1 clrlcgce02
clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1

62 1136276129 1 4 1 2 0 1 5400 -1 1
U2022S1 G1 X1 Q7 1 clrlcgce02 clrlcgce02 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1

63 1136276271 1 6 1 2 0 1 5400 -1 1
U1018S0 G1 X1 Q6 1 clrlcgce01 clrlcgce01 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1

64 1136276453 2 886 1 206 952520 1 31980
614400 5 U2003S1 G3 X1 Q5 1 clrlcgce02
clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1

65 1136276459 1 677 1 177 949400 1 31980
614400 5 U2003S1 G3 X1 Q5 1 clrlcgce02
clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1

66 1136276459 1 749 1 250 947544 1 31980
614400 5 U2003S1 G3 X1 Q5 1 clrlcgce02
clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1

67	1136276459	1	749	1	247	952104		1	31980	
	614400	5	U2003S1		G3	X1	Q5	1	clrlcgce02	
	clrlcgce02	-1	-1	-1	-1	-1	-1	-1	-1	-1
	-1	-1								
68	1136276465	0	709	1	193	947276		1	31980	
	614400	5	U2003S1		G3	X1	Q5	1	clrlcgce02	
	clrlcgce02	-1	-1	-1	-1	-1	-1	-1	-1	-1
	-1	-1								
69	1136276465	0	815	1	264	950812		1	31980	
	614400	5	U2003S1		G3	X1	Q5	1	clrlcgce02	
	clrlcgce02	-1	-1	-1	-1	-1	-1	-1	-1	-1
	-1	-1								
70	1136276465	0	815	1	251	951184		1	31980	
	614400	5	U2003S1		G3	X1	Q5	1	clrlcgce02	
	clrlcgce02	-1	-1	-1	-1	-1	-1	-1	-1	-1
	-1	-1								
71	1136276469	1	667	1	170	938420		1	31980	
	614400	5	U2003S1		G3	X1	Q5	1	clrlcgce02	
	clrlcgce02	-1	-1	-1	-1	-1	-1	-1	-1	-1
	-1	-1								
72	1136276469	1	810	1	263	965428		1	31980	
	614400	5	U2003S1		G3	X1	Q5	1	clrlcgce02	
	clrlcgce02	-1	-1	-1	-1	-1	-1	-1	-1	-1
	-1	-1								
73	1136276469	1	810	1	243	958272		1	31980	
	614400	5	U2003S1		G3	X1	Q5	1	clrlcgce02	
	clrlcgce02	-1	-1	-1	-1	-1	-1	-1	-1	-1
	-1	-1								
74	1136276474	1	923	1	260	956108		1	31980	
	614400	5	U2003S1		G3	X1	Q5	1	clrlcgce02	
	clrlcgce02	-1	-1	-1	-1	-1	-1	-1	-1	-1
	-1	-1								

75 1136276474 1 823 1 162 916344 1 31980 614400 5 U2003S1 G3 X1 Q5 1 clrlcgce02 clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

76 1136276479 1 403 1 199 951268 1 31980 614400 5 U2003S1 G3 X1 Q5 1 clrlcgce02 clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

77 1136276479 2 785 1 259 955508 1 31980 614400 5 U2003S1 G3 X1 Q5 1 clrlcgce02 clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

78 1136276479 1 681 1 170 945340 1 31980 614400 5 U2003S1 G3 X1 Q5 1 clrlcgce02 clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

79 1136276516 2 620 1 173 944720 1 31980 614400 5 U2003S1 G3 X1 Q5 1 clrlcgce02 clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

80 1136276516 1 480 1 237 966640 1 31980 614400 5 U2003S1 G3 X1 Q5 1 clrlcgce02 clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

81 1136276517 0 726 1 273 954628 1 31980 614400 5 U2003S1 G3 X1 Q5 1 clrlcgce02 clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

82 1136276518 1 724 1 243 951976 1 31980 614400 5 U2003S1 G3 X1 Q5 1 clrlcgce02 clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

83	1136276518	1	690	1	237	946980		1	31980	
	614400	5	U2003S1		G3	X1	Q5	1	clrlcgce02	
	clrlcgce02	-1	-1	-1	-1	-1	-1	-1	-1	-1
	-1	-1								
84	1136276521	2	720	1	266	956664		1	31980	
	614400	5	U2003S1		G3	X1	Q5	1	clrlcgce02	
	clrlcgce02	-1	-1	-1	-1	-1	-1	-1	-1	-1
	-1	-1								
85	1136276522	1	686	1	247	952228		1	31980	
	614400	5	U2003S1		G3	X1	Q5	1	clrlcgce02	
	clrlcgce02	-1	-1	-1	-1	-1	-1	-1	-1	-1
	-1	-1								
86	1136276522	1	720	1	267	950800		1	31980	
	614400	5	U2003S1		G3	X1	Q5	1	clrlcgce02	
	clrlcgce02	-1	-1	-1	-1	-1	-1	-1	-1	-1
	-1	-1								
87	1136276523	1	438	1	209	944552		1	31980	
	614400	5	U2003S1		G3	X1	Q5	1	clrlcgce02	
	clrlcgce02	-1	-1	-1	-1	-1	-1	-1	-1	-1
	-1	-1								
88	1136276523	1	685	1	208	945992		1	31980	
	614400	5	U2003S1		G3	X1	Q5	1	clrlcgce02	
	clrlcgce02	-1	-1	-1	-1	-1	-1	-1	-1	-1
	-1	-1								
89	1136276771	1	1856	1	8	59652	1	31980	614400	
	1	U2003S1		G3	X1	Q5	1	clrlcgce02	clrlcgce02	
	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
90	1136276771	0	1853	1	8	57124	1	31980	614400	
	1	U2003S1		G3	X1	Q5	1	clrlcgce02	clrlcgce02	
	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
91	1136276771	0	1855	1	9	59664	1	31980	614400	
	1	U2003S1		G3	X1	Q5	1	clrlcgce02	clrlcgce02	
	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

92 1136276772 1 436 1 208 962728 1 31980 614400 5 U2003S1 G3 X1 Q5 1 clrlcgce02 clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

93 1136276773 1 1856 1 9 57172 1 31980 614400 1 U2003S1 G3 X1 Q5 1 clrlcgce02 clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

94 1136276997 1 413 1 210 953524 1 31980 614400 5 U2003S1 G3 X1 Q5 1 clrlcgce02 clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

95 1136276997 1 448 1 226 958916 1 31980 614400 5 U2003S1 G3 X1 Q5 1 clrlcgce02 clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

96 1136276998 1 1856 1 9 86104 1 31980 614400 1 U2003S1 G3 X1 Q5 1 clrlcgce02 clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

97 1136277000 1 1850 1 9 57152 1 31980 614400 1 U2003S1 G3 X1 Q5 1 clrlcgce02 clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

98 1136277000 1 1854 1 10 66828 1 31980 614400 1 U2003S1 G3 X1 Q5 1 clrlcgce02 clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

99 1136277003 0 1850 1 9 57172 1 31980 614400 1 U2003S1 G3 X1 Q5 1 clrlcgce02 clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

100 1136277003 0 483 1 260 952220 1 31980 614400 5 U2003S1 G3 X1 Q5 1 clrlcgce02 clrlcgce02 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

APPENDIX C

Comparison of Job Scheduling Algorithms

A total of seventeen scheduling algorithms have been compared on an experimental Grid using synthetic, LCG1 and AuverGrid workload traces. Detailed results for each experiment under dynamic Grid scheduling environment are shown in Table C.1 to Table C.42.

Baseline Approaches	**Proposed Scheduling Algorithms**
1. First Come First Served (FCFS)	1. Multilevel Hybrid scheduling algorithms (MH)
2. Shorted Process Next (SPN)	2. Multilevel Dual Queue Scheduling algorithms (MDQ)
3. Longest Job First (LJF)	3. Dynamic Multilevel Hybrid Scheduling Algorithm using Median(MHM)
4. Priority(P)	4. Dynamic Multilevel Hybrid Scheduling Algorithm using square root(MHR)
5. Round Robin (RR)	5. Dynamic Multilevel Dual Queue Scheduling Algorithm using Median (MDQM)
6. Proportional Local Round Robin (PLRR)	6. Dynamic Multilevel Dual Queue Scheduling Algorithm using Square root (MDQR)
7. Self Adjustment Round Robin (SARR)	
8. Intelligent Time slice for Round Robin (NIR)	
9. Round Robin Priority (NRR)	
10. Multilevel CPU Scheduling algorithm(MR)	
11. Shortest Remaining Burst Round Robin (SRBRR)	

Table C.1: Average waiting times (seconds) of scheduling algorithms for synthetic workload of 1000 processes

Scheduling Algorithms	**Number of CPUs**		
	16	**32**	**64**
FCFS	69410.11	31858.33	13413.73
LJF	112096.6	51320.24	21331.94
SPN	24504.02	11851.98	5965.493
P	66474.7	30353.9	13084.7
RR	46601.48	21324.58	9073.374
PLRR	66937.62	30082.4	13172.98
MH	22372.98	10045.66	4188.51
MHM	29682.45	14262	6939.979
MHR	22429.11	10074.03	4203.508
MDQ	45570.29	21136.17	9031.878
MDQM	30441.99	15702.45	7897.8
MDQR	45611.46	21140.16	9044.48
SARR	55887.71	26049.61	10983.42
NIR	46601.47	21325.36	9069.637
NRR	68021.86	27293.15	9663.937
MR	24504.02	11851.98	5965.493
SRBRR	51206.63	22405.31	10038.4

Table C.2: Average waiting times (seconds) of scheduling algorithms for synthetic workload of 2000 processes

Scheduling Algorithms	**Number of CPUs**		
	16	**32**	**64**
FCFS	136016.3	62657.16	26297.68
LJF	224014	104468.5	44909.67
SPN	43285.85	20129.72	8946.296
P	132158.2	61022.03	25870.3
RR	87417.27	40215.56	16795.34
PLRR	130268.6	58864.64	25682.06
MH	41573.64	18631.26	7429.109
MHM	53936.86	25269.46	11078.94
MHR	41666.55	18680.53	7456.65
MDQ	84332.47	39494.8	16633.61
MDQM	55201.16	27313.16	13009.66
MDQR	84244.91	39525.37	16650.96
SARR	102536	49411.62	20479.99
NIR	87432.7	40217.64	16792.5
NRR	135457.5	60372.33	21642.47
MR	43285.85	20129.72	8946.296
SRBRR	93473.64	44436.3	18843.73

Table C.3: Average waiting times (seconds) of scheduling algorithms for ‘10%’ workload of LCG1

Scheduling Algorithms	**Number of CPUs**			
	16	**32**	**64**	**128**
FCFS	2861611	1398768	667700	305900.6
LJF	5950838	2931544	1407113	600534.8
SPN	323441	146513.1	61139.94	30715.23
P	3082319	1499720	701431.9	294508.7
RR	674747	311521.3	134544.8	54301.88
PLRR	1690220	840228.5	409916.6	49143.2
MH	321597	144667.8	59496.62	23490.8
MHM	352893	161483.1	68453.1	33821.22
MHR	321822	144795.6	59578.99	23557.06
MDQ	659884	307613.4	133811.1	54209.95
MDQM	416628	209752.5	99178.06	52891.19
MDQR	643433	303071.8	132637.4	53971.32
SARR	767175	373983.3	176500.7	55602.99
NIR	674712	311475.1	134560.7	54296.82
NRR	2861461	1398768	667700	542306
MR	323425	146513.1	61139.94	43543.86
SRBRR	656984	323416.7	135262.3	68531.33

Table C.4: Average waiting times (seconds) of scheduling algorithms for ‘20%’ workload of LCG1

Scheduling Algorithms	Number of CPUs			
	16	**32**	**64**	**128**
FCFS	7234327.7	3560391	1723600	809300.1
LJF	1.35E+07	6688859	3256909	1487380
SPN	721365.83	333266.9	145040.7	65122.78
P	7253238.2	3559867	1701670	766741.9
RR	1501999.7	707696.1	318016.7	135434.6
PLRR	4061030.9	2017145	1006314	585555.9
MH	713273.67	331741.2	143640	59210.78
MHM	775951.11	362058.5	159998.8	72404.8
MHR	720114.76	331928.1	143763.8	59306.99
MDQ	1468445.5	699270	316134.3	135179.3
MDQM	879428.59	448064.5	218005.2	114052.9
MDQR	1427977.4	685619.9	312595.2	134391.7
SARR	1632564.1	859121.4	454805.5	136747.2
NIR	1501856.8	707678.4	318021.7	135438.3
NRR	7234327.7	3560391	1723600	801313.4
MR	721365.83	333266.9	145040.7	65122.78
SRBRR	1361201.8	691006.7	309453.9	156396.4

Table C.5: Average waiting times (seconds) of scheduling algorithms for ‘3%’ workload of AuverGrid

Scheduling Algorithms	Number of CPUs			
	16	**32**	**64**	**128**
FCFS	5990176	2293049	471976.7	311504.6
LJF	7936983	3788116	802360.4	523327.8
SPN	4031367	1411401	234475.3	33995.62
P	5735163	2158118	426195.5	33472.82
RR	2804801	1080765	211061.6	8766.438
PLRR	5905104	2088255	418870.9	31743.93
MH	1194281	418123.7	69462.65	4278.066
MHM	1488958	553338.9	69462.65	14738.8
MHR	1196159	419250.4	69921.53	4215.228
MDQ	2781283	1075495	210723.1	8764.153
MDQM	1494964	594775.2	161109.8	17228.26
MDQR	2714622	1061980	210064.9	8792.504
SARR	3114754	1081584	211202.3	9221.532
NIR	2804578	1080691	211041.1	8766.035
NRR	5919852	2259157	429147.2	23839.38
MR	1500808	648090	130321.5	8375.194
SRBRR	2925552	1353006	301670.1	15475.23

Table C.6: Average waiting times (seconds) of scheduling algorithms for ‘5%’ workload of AuverGrid

Scheduling Algorithms	Number of CPUs			
	16	**32**	**64**	**128**
FCFS	20366598	7911017	1662774	1003045
LJF	26588893	12235615	2591624	1674649
SPN	4031367	1411401	234475.3	33995.62
P	19614256	7380762	1457589	114477.1
RR	9620467	3707025	723941.4	30068.88
PLRR	20136405	7120949	1428350	108246.8
MH	3726158	1304546	216723.5	13347.57
MHM	4749777	1765151	221585.9	47016.78
MHR	3720055	1303869	217456	13109.36
MDQ	8927919	3452339	676421.2	28132.93
MDQM	4828733	1921124	520384.6	55647.27
MDQR	8768229	3430197	678509.5	28399.79
SARR	10060654	3493517	682183.3	29785.55
NIR	9872113	3804031	742864.7	30856.44
NRR	20482689	7816685	1484849	82484.25
MR	5177788	2235910	449609.2	28894.42
SRBRR	9478788	4383740	977411.3	50139.74

Table C.7: Average turnaround times (seconds) of scheduling algorithms for synthetic workload of 1000 processes

Scheduling Algorithms	**Number of CPUs**		
	16	**32**	**64**
FCFS	71773.55	34221.77	15777.17
LJF	114460	53683.67	23695.38
SPN	26867.45	14215.42	8328.93
P	68838.14	32717.33	15448.14
RR	48964.92	23688.02	11436.81
PLRR	69301.06	32445.83	15536.42
MH	24736.41	12409.1	6551.947
MHM	32045.89	16625.44	9303.416
MHR	24792.55	12437.47	6566.945
MDQ	47933.73	23499.61	11395.32
MDQM	32805.42	18065.89	10261.24
MDQR	47974.9	23503.6	11407.92
SARR	58251.15	28413.05	13346.86
NIR	48964.9	23688.79	11433.07
NRR	70385.3	29656.58	12027.37
MR	26867.45	14215.42	8328.93
SRBRR	53570.06	24768.75	12401.83

Table C.8: Average turnaround times (seconds) of scheduling algorithms for synthetic workload of 2000 processes

Scheduling Algorithms	Number of CPUs		
	16	**32**	**64**
FCFS	138316.5	64957.39	28597.9
LJF	226314.2	106768.7	47209.89
SPN	45586.08	22429.94	11246.52
P	134458.4	63322.25	28170.52
RR	89717.49	42515.78	19095.56
PLRR	132568.8	61164.86	27982.28
MH	43873.86	20931.48	9729.33
MHM	56237.08	27569.68	13379.16
MHR	43966.77	20980.75	9756.871
MDQ	86632.69	41795.02	18933.83
MDQM	57501.38	29613.38	15309.88
MDQR	86545.13	41825.59	18951.18
SARR	104836.2	51711.84	22780.21
NIR	89732.92	42517.86	19092.73
NRR	137757.7	62672.55	23942.69
MR	45586.08	22429.94	11246.52
SRBRR	95773.87	46736.52	21143.96

Table C.9: Average turnaround times (seconds) of scheduling algorithms for '10%' workload of LCG1

Scheduling Algorithms	Number of CPUs			
	16	**32**	**64**	**128**
FCFS	2867060	1404215	673147.9	311348.5
LJF	5956286	2936992	1412561	605982.6
SPN	328889	151961	66587.78	36163.07
P	3087767	1505168	706879.8	299956.5
RR	680195	316969.2	139992.7	59749.72
PLRR	1695668	845676.4	415364.4	54073.5
MH	327045	150115.7	64944.46	28938.64
MHM	358341	166930.9	73900.94	39269.06
MHR	327270	150243.4	65026.83	29004.9
MDQ	665332	313061.2	139258.9	59657.79
MDQM	422076	215200.3	104625.9	58339.03
MDQR	648881	308519.7	138085.3	59419.16
SARR	772623	379431.1	181948.5	61050.83
NIR	680160	316923	140008.5	59744.66
NRR	2866909	1404215	673147.9	546730.7
MR	328873	151961	66587.78	47423.81
SRBRR	662432	328864.5	140710.2	73979.17

Table C.10: Average turnaround times (seconds) of scheduling algorithms for '20%' workload of LCG1

Scheduling Algorithms	Number of CPUs			
	16	**32**	**64**	**128**
FCFS	7240480.96	3566544	1729753	815453.4
LJF	1.35E+07	6695013	3263062	1493533
SPN	727519.129	339420.2	151194	71276.08
P	7259391.49	3566021	1707824	772895.2
RR	1508152.97	713849.4	324170	141587.9
PLRR	4067184.17	2023298	1012467	591709.2
MH	719368.513	337894.5	149793.3	65364.08
MHM	782104.41	368211.8	166152.1	78558.1
MHR	726268.06	338081.4	149917.1	65460.29
MDQ	1474598.8	705423.3	322287.6	141332.6
MDQM	885581.897	454217.8	224158.5	120206.3
MDQR	1434130.68	691773.2	318748.5	140545
SARR	1638717.4	865274.7	460958.8	142900.5
NIR	1508010.14	713831.7	324175	141591.6
NRR	7240480.96	3566544	1729753	807466.7
MR	727519.129	339420.2	151194	71276.08
SRBRR	1367355.14	697160	315607.2	162549.7

Table C.11: Average turnaround times (seconds) of scheduling algorithms for '3%' workload of AuverGrid

Scheduling Algorithms	Number of CPUs			
	16	**32**	**64**	**128**
FCFS	6009804	2312677	491604.9	324459.2
LJF	7962990	3820542	835728.3	545091.5
SPN	4097623	1477657	300731.4	121979.6
P	5754791	2177746	445823.7	53101.01
RR	2824429	1100393	230689.8	28394.63
PLRR	5924732	2107883	438499.1	51372.12
MH	1213909	437751.9	89090.84	15350.12
MHM	1508587	572967.1	89090.84	34366.99
MHR	1215787	438878.6	89549.72	15412.96
MDQ	2800911	1095123	230351.3	28392.34
MDQM	1514592	614403.4	180738	36856.45
MDQR	2734250	1081609	229693.1	28420.69
SARR	3134382	1101212	230830.4	28849.72
NIR	2824206	1100319	230669.3	28394.22
NRR	5939481	2278786	448775.4	43467.56
MR	1510877	657491.9	138800.9	91427.77
SRBRR	2945180	1372634	321298.3	168935.3

Table C.12: Average turnaround times (seconds) of scheduling algorithms for '5%' workload of AuverGrid

Scheduling Algorithms	Number of CPUs			
	16	**32**	**64**	**128**
FCFS	20433334	7978735	1731924	1044759
LJF	26676018	12340351	2699403	1744293
SPN	4097623	1477657	300731.4	121979.6
P	19681384	7447890	1524717	181605.5
RR	9687792	3774349	791266	97393.57
PLRR	20203337	7187881	1495282	175178.9
MH	3787398	1365786	277963.4	47892.38
MHM	4812391	1827765	284199.8	109630.7
MHR	3781099	1364913	278499.6	47934.3
MDQ	8990925	3515345	739427.7	91139.41
MDQM	4892132	1984523	583783.7	119046.3
MDQR	8831629	3493596	741908.6	91798.83
SARR	10124054	3556916	745582.3	93184.59
NIR	9941204	3873122	811955.9	99947.67
NRR	20550603	7884598	1552763	150397.8
MR	5212527	2268347	478863	315425.8
SRBRR	9542383	4447335	1041007	547350.3

Table C.13: Average response times (seconds) of scheduling algorithms for synthetic workload of 1000 processes

Scheduling Algorithms	**Number of CPUs**		
	16	**32**	**64**
FCFS	69410.11	31858.33	13413.73
LJF	112096.6	51320.24	21331.94
SPN	24504.02	11851.98	5965.493
P	66474.7	30353.9	13084.7
RR	570.572	221.415	81.626
PLRR	62709.29	27564.91	12484.49
MH	18562.18	7278.342	2281.433
MHM	26810.76	13061.88	6405.51
MHR	18602.23	7313.973	2288.173
MDQ	1131.018	421.429	148.01
MDQM	22207.23	11143.99	5930.715
MDQR	1211.761	464.395	175.771
SARR	26216.1	13261.16	5711.544
NIR	563.153	226.32	82.772
NRR	60549.46	18154.49	2963.13
MR	24504.02	11851.98	5965.493
SRBRR	19794.67	10015.67	5471.248

Table C.14: Average response times (seconds) of scheduling algorithms for synthetic workload of 2000 processes

Scheduling Algorithms	Number of CPUs		
	16	**32**	**64**
FCFS	136016.3	62657.16	26297.68
LJF	224014	104468.5	44909.67
SPN	43285.85	20129.72	8946.296
P	132158.2	61022.03	25870.3
RR	1133.856	454.1655	175.214
PLRR	121696	53524.27	24587.66
MH	35526.22	14382.46	4491.348
MHM	48188.73	22456.12	9906.094
MHR	35594.07	14435.71	4497.006
MDQ	2311.941	853.698	307.0905
MDQM	44742	20090.68	8858.263
MDQR	2415.802	954.9475	360.2245
SARR	47375.03	25127.37	10390.58
NIR	1134.045	454.5745	178.9355
NRR	133743.8	54924.55	13906.56
MR	43285.85	20129.72	8946.296
SRBRR	38979.22	19234.34	8261.744

Table C.15: Average response times (seconds) of scheduling algorithms for ‘10%’ workload of LCG1

Scheduling Algorithms	Number of CPUs			
	16	**32**	**64**	**128**
FCFS	2861611	1398768	667700	305900.6
LJF	5950838	2931544	1407113	600534.8
SPN	323441	146513.1	61139.94	30715.23
P	3082319	1499720	701431.9	294508.7
RR	10228.1	4094.132	1425.504	450.5494
PLRR	1479992	747505.6	368478.2	407.7472
MH	311632	137110.4	52955.72	18944.28
MHM	324531	146214.8	61145.56	31013.72
MHR	311659	137169.9	52980.61	18998.3
MDQ	21620.8	8048.322	2567.438	718.2185
MDQM	278263	80734.89	31145.9	21268.49
MDQR	32521.6	13470.59	5027.868	1704.814
SARR	241779	134866.3	49683.54	1635.479
NIR	10674.2	4063.979	1436.445	449.3505
NRR	2861461	1398768	667700	542306
MR	323425	146513.1	61139.94	43543.86
SRBRR	117714	57997.54	17013.07	19493.14

Table C.16: Average response times (seconds) of scheduling algorithms for ‘20%’ workload of LCG1

Scheduling Algorithms	Number of CPUs			
	16	**32**	**64**	**128**
FCFS	7234327.659	3560391	1723600	809300.1
LJF	1.35E+07	6688859	3256909	1487380
SPN	721365.8251	333266.9	145040.7	65122.78
P	7253238.182	3559867	1701670	766741.9
RR	17542.03345	8829.081	3322.127	1096.324
PLRR	3570384.236	1791130	904793.3	556276
MH	703884.9578	323833.9	136052.4	52747.63
MHM	750990.8151	346664.1	150966	67208.89
MHR	710635.9998	323609.3	136153.2	52860.07
MDQ	45679.90693	17821.97	6157.432	1805.267
MDQM	610903.7537	175045.7	66112.59	38830.31
MDQR	76498.2123	32106.37	12609.67	4425.426
SARR	413957.9315	298812.7	141822.2	1808.586
NIR	16877.0322	8650.061	3335.242	1097.624
NRR	7234327.659	3560391	1723600	800190.3
MR	721365.8251	333266.9	145040.7	65122.78
SRBRR	219541.7004	117339.5	36337.2	34905.39

Table C.17: Average response times (seconds) of scheduling algorithms for ‘3%’ workload of AuverGrid

Scheduling Algorithms	Number of CPUs			
	16	**32**	**64**	**128**
FCFS	5990176	2293049	471976.7	311504.6
LJF	7936983	3788116	802360.4	523327.8
SPN	3791641	1215306	130041.3	77663.49
P	5735163	2158118	426195.5	33472.82
RR	915263.2	616040	405064.4	228469.2
PLRR	1582077	554561.1	107374	192055.4
MH	1123263	360031	38524.36	9773.303
MHM	1414622	516635.1	38524.36	13666.01
MHR	1122509	351919.8	36164.43	9698.071
MDQ	8309.096	12273.83	13992.12	14398.55
MDQM	1272995	432891.5	98085.13	12751.02
MDQR	9575.527	5928.119	12694.07	14271.58
SARR	723762.4	614541.8	404763.4	227955.6
NIR	915254.9	615835.8	405081.4	228450.8
NRR	1627154	779068	104362.8	204914
MR	340881.4	182980.9	145811.9	97376.53
SRBRR	664486.1	382006	337527.6	179927.1

Table C.18: Average response times (seconds) of scheduling algorithms for ‘5%’ workload of AuverGrid

Scheduling Algorithms	Number of CPUs			
	16	**32**	**64**	**128**
FCFS	20366598	7911017	1662774	1003045
LJF	26588893	12235615	2591624	1674649
SPN	3791641	1215306	130041.3	77663.49
P	19614256	7380762	1457589	114477.1
RR	3139353	2113017	1389371	783649.2
PLRR	5394883	1891053	366145.2	654909
MH	3504581	1123297	120196	30492.7
MHM	4512644	1648066	122892.7	43594.56
MHR	3491002	1094470	112471.4	30161
MDQ	26672.2	39398.99	44914.72	46219.34
MDQM	4111774	1398240	316815	41185.78
MDQR	30928.95	19147.82	41001.86	46097.22
SARR	2337753	1984970	1307386	736296.6
NIR	3221697	2167742	1425887	804146.9
NRR	5629954	2695575	361095.4	709002.4
MR	1176041	631284	503051.1	335949
SRBRR	2152935	1237699	1093589	582963.7

Table C.19: Average slowdown times (seconds) of scheduling algorithms for synthetic workload of 1000 processes

Scheduling Algorithms	**Number of CPUs**		
	16	**32**	**64**
FCFS	251.167	114.807	48.024
LJF	409.796	189.403	79.875
SPN	90.919	45.914	25.68
P	237.217	109.187	47.401
RR	25.529	10.316	4.32
PLRR	227.3	98.349	45.2
MH	4.837	2.492	1.484
MHM	16.095	11.957	10.888
MHR	4.852	2.5	1.49
MDQ	24.077	10.211	4.309
MDQM	23.44	19.76	14.216
MDQR	24.249	10.362	4.388
SARR	105.049	51.183	21.913
NIR	25.55	10.344	4.318
NRR	221.295	67.575	12.212
MR	90.919	45.914	25.68
SRBRR	170.801	72.91	31.803

Table C.20: Average slowdown times (seconds) of scheduling algorithms for synthetic workload of 2000 processes

Scheduling Algorithms	Number of CPUs		
	16	**32**	**64**
FCFS	482.0865	220.423	91.831
LJF	797.325	370.162	160.6275
SPN	157.9445	75.1945	36.225
P	468.8215	215.6475	91.8645
RR	50.7945	20.352	8.094
PLRR	430.128	189.286	86.919
MH	8.3775	3.945	1.9725
MHM	19.6405	13.189	10.6325
MHR	8.4055	3.96	1.9825
MDQ	46.0235	19.698	8.027
MDQM	35.887	26.4775	17.5925
MDQR	46.4985	20.0065	8.161
SARR	179.796	93.618	38.8185
NIR	50.881	20.397	8.1085
NRR	474.035	193.443	49.4715
MR	157.9445	75.1945	36.225
SRBRR	292.163	133.8875	54.268

Table C.21: Average slowdown times (seconds) of scheduling algorithms for '10%' workload of LCG1

Scheduling Algorithms	Number of CPUs			
	16	**32**	**64**	**128**
FCFS	13117.9	6421.734	3064.842	1402.675
LJF	27462.7	13528.99	6493.688	2719.321
SPN	1531.28	704.8904	306.9519	167.5365
P	14292.6	6946.97	3257.16	1344.321
RR	461.224	173.932	58.80509	18.31594
PLRR	6776.4	3399.734	1678.194	16.57593
MH	55.8087	18.3688	4.426133	1.793395
MHM	62.0095	23.49495	10.20666	34.7993
MHR	55.8753	18.43656	4.524037	1.907732
MDQ	427.365	170.4546	58.87226	18.49346
MDQM	388.624	214.0648	103.0869	81.68406
MDQR	394.433	164.8052	59.13556	18.77654
SARR	1236.3	678.3175	288.2983	25.16369
NIR	461.659	173.6755	58.83833	18.30988
NRR	13117.2	6421.734	3064.842	2489.265
MR	1531.2	704.8904	306.9519	218.6111
SRBRR	2735.72	1129.372	279.9713	138.9497

Table C.22: Average slowdown times (seconds) of scheduling algorithms for '20%' workload of LCG1

Scheduling Algorithms	Number of CPUs			
	16	**32**	**64**	**128**
FCFS	34178.7	16842.9	8150.154	3829.503
LJF	62418.4	30879.95	15011.28	6812.01
SPN	3694.86	1727.869	760.8359	349.5565
P	33683.9	16533.99	7903.295	3536.74
RR	977.131	385.4179	137.728	44.34272
PLRR	17050.2	8500.956	4273.18	2625.981
MH	108.67	33.37375	8.983913	3.255478
MHM	115.478	37.50819	13.82331	28.25893
MHR	109.712	33.47426	9.103435	3.396937
MDQ	896.593	376.4444	138.0286	45.04669
MDQM	640.896	400.5188	211.1163	145.3005
MDQR	821.247	358.3647	136.7946	45.81453
SARR	2174.49	1535.133	880.2057	51.14843
NIR	975.856	385.0248	137.8052	44.35649
NRR	34178.7	16842.9	8150.154	3785.545
MR	3694.86	1727.869	760.8359	349.5565
SRBRR	6008.45	2592.881	711.1455	380.2265

Table C.23: Average slowdown times (seconds) of scheduling algorithms for ‘3%’ workload of AuverGrid

Scheduling Algorithms	Number of CPUs			
	16	**32**	**64**	**128**
FCFS	39192.75	15014.18	3124.658	2062.274
LJF	51930.4	24803.43	5311.918	3464.62
SPN	53.53066	18.66844	5.903679	8.345621
P	35194.33	13348.59	2763.564	329.015
RR	137.39	50.05443	10.39208	1.623835
PLRR	37737.99	12813.94	2675.489	291.6865
MH	15.85831	5.530474	1.748948	1.050227
MHM	154.8647	223.1314	1.748948	159.7599
MHR	16.35085	6.081485	2.272907	1.253691
MDQ	135.1559	49.42796	10.37798	1.629113
MDQM	651.6212	605.7148	424.6117	161.7068
MDQR	187.1867	70.75695	14.85897	2.050309
SARR	4421.499	60.28619	12.66202	5.276619
NIR	137.5692	50.08107	10.34359	1.622268
NRR	38261.43	14648.85	2644.199	185.32
MR	6090.147	2525.244	451.8363	102295.2
SRBRR	11871.63	5271.908	1045.917	189015.5

Table C.24: Average slowdown times (seconds) of scheduling algorithms for ‘5%’ workload of AuverGrid

Scheduling Algorithms	Number of CPUs			
	16	**32**	**64**	**128**
FCFS	133255.4	51798.93	11008.17	6640.522
LJF	173966.8	80115.08	17157.49	11086.78
SPN	53.53066	18.66844	5.903679	8.345621
P	120364.6	45652.19	9451.389	1125.231
RR	471.2478	171.6867	35.64484	5.569754
PLRR	128686.6	43695.52	9123.418	994.651
MH	49.47792	17.25508	5.456719	3.276708
MHM	494.0183	711.7891	5.579146	509.6341
MHR	50.85113	18.91342	7.068741	3.898978
MDQ	433.8504	158.6637	33.31331	5.229454
MDQM	2104.736	1956.459	1371.496	522.313
MDQR	604.6131	228.5449	47.99447	6.622499
SARR	14281.44	194.7244	40.89833	17.04348
NIR	484.2434	176.2854	36.40943	5.710384
NRR	132384.6	50685.02	9148.929	641.2072
MR	21011.01	8712.091	1558.835	352918.3
SRBRR	38464.09	17080.98	3388.772	612410.1

Table C.25: Total completion times (seconds) of scheduling algorithms for synthetic workload of 1000 processes

Scheduling Algorithms	Number of CPUs		
	16	**32**	**64**
FCFS	7.68E+07	3.92E+07	2.08E+07
LJF	1.19E+08	5.87E+07	2.87E+07
SPN	3.19E+07	1.92E+07	1.33E+07
P	7.38E+07	3.77E+07	2.04E+07
RR	5.40E+07	2.87E+07	1.64E+07
PLRR	7.43E+07	3.74E+07	2.05E+07
MH	2.97E+07	1.74E+07	1.15E+07
MHM	3.70E+07	2.16E+07	1.43E+07
MHR	2.98E+07	1.74E+07	1.16E+07
MDQ	5.29E+07	2.85E+07	1.64E+07
MDQM	3.78E+07	2.31E+07	1.53E+07
MDQR	5.30E+07	2.85E+07	1.64E+07
SARR	6.32E+07	3.34E+07	1.83E+07
NIR	5.40E+07	2.87E+07	1.64E+07
NRR	7.54E+07	3.47E+07	1.70E+07
MR	3.19E+07	1.92E+07	1.33E+07
SRBRR	5.86E+07	2.98E+07	1.74E+07

Table C.26: Total completion times (seconds) of scheduling algorithms for synthetic workload of 2000 processes

Scheduling Algorithms	Number of CPUs		
	16	**32**	**64**
FCFS	2.97E+08	1.50E+08	7.72E+07
LJF	4.73E+08	2.34E+08	1.14E+08
SPN	1.11E+08	6.49E+07	4.25E+07
P	2.89E+08	1.47E+08	7.63E+07
RR	1.99E+08	1.05E+08	5.82E+07
PLRR	2.85E+08	1.42E+08	7.60E+07
MH	1.08E+08	6.19E+07	3.95E+07
MHM	1.32E+08	7.51E+07	4.68E+07
MHR	1.08E+08	6.20E+07	3.95E+07
MDQ	1.93E+08	1.04E+08	5.79E+07
MDQM	1.35E+08	7.92E+07	5.06E+07
MDQR	1.93E+08	1.04E+08	5.79E+07
SARR	2.30E+08	1.23E+08	6.56E+07
NIR	1.99E+08	1.05E+08	5.82E+07
NRR	2.96E+08	1.45E+08	6.79E+07
MR	1.11E+08	6.49E+07	4.25E+07
SRBRR	2.12E+08	1.13E+08	6.23E+07

Table C.27: Total completion times (seconds) of scheduling algorithms for ‘10%’ workload of LCG1

Scheduling Algorithms	Number of CPUs			
	16	**32**	**64**	**128**
FCFS	5.51E+10	2.76E+10	1.38E+10	7.01E+09
LJF	1.13E+11	5.64E+10	2.77E+10	1.25E+10
SPN	7.34E+09	4.01E+09	2.41E+09	1.83E+09
P	5.92E+10	2.95E+10	1.44E+10	6.79E+09
RR	1.39E+10	7.11E+09	3.79E+09	2.28E+09
PLRR	3.30E+10	1.71E+10	8.96E+09	2.06E+09
MH	7.30E+09	3.98E+09	2.37E+09	1.70E+09
MHM	7.89E+09	4.29E+09	2.54E+09	1.89E+09
MHR	7.31E+09	3.98E+09	2.38E+09	1.70E+09
MDQ	1.37E+10	7.04E+09	3.77E+09	2.28E+09
MDQM	9.09E+09	5.20E+09	3.12E+09	2.25E+09
MDQR	1.34E+10	6.95E+09	3.75E+09	2.27E+09
SARR	1.57E+10	8.29E+09	4.57E+09	2.30E+09
NIR	1.39E+10	7.11E+09	3.79E+09	2.28E+09
NRR	5.51E+10	2.76E+10	1.38E+10	1.12E+10
MR	7.34E+09	4.01E+09	2.41E+09	1.71E+09
SRBRR	1.36E+10	7.34E+09	3.80E+09	2.54E+09

Table C.28: Total completion times (seconds) of scheduling algorithms for ‘20%’ workload of LCG1

Scheduling Algorithms	Number of CPUs			
	16	**32**	**64**	**128**
FCFS	2.76E+11	1.38E+11	6.92E+10	3.48E+10
LJF	5.13E+11	2.56E+11	1.27E+11	6.03E+10
SPN	3.15E+10	1.69E+10	9.85E+09	6.84E+09
P	2.77E+11	1.38E+11	6.84E+10	3.32E+10
RR	6.09E+10	3.10E+10	1.64E+10	9.48E+09
PLRR	1.57E+11	8.03E+10	4.22E+10	2.64E+10
MH	3.12E+10	1.69E+10	9.79E+09	6.62E+09
MHM	3.36E+10	1.80E+10	1.04E+10	7.11E+09
MHR	3.15E+10	1.69E+10	9.80E+09	6.62E+09
MDQ	5.96E+10	3.07E+10	1.63E+10	9.48E+09
MDQM	3.75E+10	2.12E+10	1.26E+10	8.68E+09
MDQR	5.81E+10	3.02E+10	1.61E+10	9.45E+09
SARR	6.58E+10	3.67E+10	2.15E+10	9.53E+09
NIR	6.09E+10	3.10E+10	1.64E+10	9.48E+09
NRR	2.76E+11	1.38E+11	6.92E+10	3.45E+10
MR	3.15E+10	1.69E+10	9.85E+09	6.84E+09
SRBRR	5.56E+10	3.04E+10	1.60E+10	1.03E+10

Table C.29: Total completion times (seconds) of scheduling algorithms for ‘3%’ workload of AuverGrid

Scheduling Algorithms	Number of CPUs			
	16	**32**	**64**	**128**
FCFS	9.23E+10	4.75E+10	2.54E+10	1.68E+10
LJF	1.22E+11	7.84E+10	4.32E+10	2.82E+10
SPN	1.15E+11	8.35E+10	6.93E+10	1.56E+11
P	8.92E+10	4.58E+10	2.48E+10	2.01E+10
RR	5.37E+10	3.28E+10	2.22E+10	1.98E+10
PLRR	9.13E+10	4.50E+10	2.48E+10	2.01E+10
MH	3.42E+10	2.47E+10	2.05E+10	1.96E+10
MHM	3.77E+10	2.64E+10	2.05E+10	1.99E+10
MHR	3.42E+10	2.48E+10	2.05E+10	1.96E+10
MDQ	5.34E+10	3.27E+10	2.22E+10	1.98E+10
MDQM	3.78E+10	2.69E+10	2.16E+10	1.99E+10
MDQR	5.26E+10	3.26E+10	2.22E+10	1.98E+10
SARR	5.74E+10	3.28E+10	2.22E+10	1.98E+10
NIR	5.37E+10	3.28E+10	2.22E+10	1.98E+10
NRR	9.15E+10	4.71E+10	2.49E+10	2.00E+10
MR	2.83E+10	1.73E+10	1.01E+10	2107101
SRBRR	5.52E+10	3.61E+10	2.33E+10	3893386

Table C.30: Total completion times (seconds) of scheduling algorithms for ‘5%’ workload of AuverGrid

Scheduling Algorithms	Number of CPUs			
	16	**32**	**64**	**128**
FCFS	3.14E+11	1.64E+11	8.95E+10	5.40E+10
LJF	4.10E+11	2.53E+11	1.39E+11	9.01E+10
SPN	1.15E+11	8.35E+10	6.93E+10	1.56E+11
P	3.05E+11	1.57E+11	8.50E+10	6.87E+10
RR	1.84E+11	1.12E+11	7.63E+10	6.79E+10
PLRR	3.11E+11	1.53E+11	8.44E+10	6.84E+10
MH	1.07E+11	7.72E+10	6.40E+10	6.12E+10
MHM	1.20E+11	8.42E+10	6.55E+10	6.33E+10
MHR	1.06E+11	7.70E+10	6.38E+10	6.10E+10
MDQ	1.71E+11	1.05E+11	7.14E+10	6.35E+10
MDQM	1.22E+11	8.69E+10	6.99E+10	6.42E+10
MDQR	1.70E+11	1.05E+11	7.18E+10	6.39E+10
SARR	1.86E+11	1.06E+11	7.18E+10	6.39E+10
NIR	1.89E+11	1.15E+11	7.83E+10	6.96E+10
NRR	3.16E+11	1.63E+11	8.61E+10	6.91E+10
MR	9.76E+10	5.96E+10	3.48E+10	7269497
SRBRR	1.79E+11	1.17E+11	7.56E+10	12614571

Table C.31: Maximum job stretch times (seconds) of scheduling algorithms for synthetic workload of 1000 processes

Scheduling Algorithms	**Number of CPUs**		
	16	**32**	**64**
FCFS	1.17E+04	5716	3464
LJF	12718	6239	3477
SPN	10332	5121	2358
P	1.26E+04	6240	3107
RR	132	56	30
PLRR	11736	5716	2631
MH	19	11	6
MHM	994	987	761
MHR	19	11	6
MDQ	121	58	26
MDQM	994	987	761
MDQR	163	65	37
SARR	6086	2971	1271
NIR	136	58	30
NRR	10927	4124	974
MR	10332	5121	2358
SRBRR	10066	5417	3046

Table C.32: Maximum job stretch times (seconds) of scheduling algorithms for synthetic workload of 2000 processes

Scheduling Algorithms	**Number of CPUs**		
	16	**32**	**64**
FCFS	2.10E+04	9641	4738
LJF	24367	12122	5564
SPN	19472	8062	4738
P	2.33E+04	12609	6595
RR	246	128	37
PLRR	20964	9641	4366
MH	31	19	10
MHM	991	986	851
MHR	31	19	11
MDQ	263	125	48
MDQM	1377	1368	851
MDQR	313	138	61
SARR	9886	4356	2722
NIR	248	129	41
NRR	20283	9379	3231
MR	19472	8062	4738
SRBRR	18659	9543	4262

Table C.33: Maximum job stretch times (seconds) of scheduling algorithms for '10%' workload of LCG1

Scheduling Algorithms	**Number of CPUs**			
	16	**32**	**64**	**128**
FCFS	3359816	1602593	761848	372911
LJF	7063845	2857339	1542261	801865
SPN	1381162	677465	280263	157647
P	2916861	1513005	629592	558689
RR	17184	7289	1990	801
PLRR	3359816	628540	433272	724.905
MH	271	145	57	50
MHM	646	1266	2014	73975
MHR	271	145	57	87
MDQ	18172	5483	2922	500
MDQM	79858	66903	30229	73975
MDQR	20532	10216	3161	1439
SARR	459190	107625	88377	16729
NIR	17609	7579	2612	957
NRR	3359816	1602593	761848	618772.9
MR	1381162	677465	280263	199603.3
SRBRR	1131862	376086	583561	95439

Table C.34: Maximum job stretch times (seconds) of scheduling algorithms for '20%' workload of LCG1

Scheduling Algorithms	Number of CPUs			
	16	**32**	**64**	**128**
FCFS	1.44E+07	6774601	3612479	1899261
LJF	1.51E+07	7529773	4255895	2466950
SPN	7599373	4094328	2070857	730452
P	1.03E+07	5357188	3046257	1592032
RR	80818	26835	7439	2843
PLRR	1.44E+07	6774601	3612479	1899261
MH	491.288	207	89	50
MHM	646	1266	2014	83999
MHR	496	207	155	190
MDQ	56019	33567	11096	3911
MDQM	176855	169186	87956	161781
MDQR	134426	37927	25632	10525
SARR	980513	1088186	626878	14397
NIR	81889	27770	6917	3351
NRR	1.44E+07	6774601	3612479	1899261
MR	7599373	4094328	2070857	730452
SRBRR	2517004	4713827	1311760	811871

Table C.35: Total completion times (seconds) of scheduling algorithms for ‘3%’ workload of AuverGrid

Scheduling Algorithms	Number of CPUs			
	16	**32**	**64**	**128**
FCFS	9.23E+10	4.75E+10	2.54E+10	1.68E+10
LJF	1.22E+11	7.84E+10	4.32E+10	2.82E+10
SPN	1.15E+11	8.35E+10	6.93E+10	1.56E+11
P	8.92E+10	4.58E+10	2.48E+10	2.01E+10
RR	5.37E+10	3.28E+10	2.22E+10	1.98E+10
PLRR	9.13E+10	4.50E+10	2.48E+10	2.01E+10
MH	3.42E+10	2.47E+10	2.05E+10	1.96E+10
MHM	3.77E+10	2.64E+10	2.05E+10	1.99E+10
MHR	3.42E+10	2.48E+10	2.05E+10	1.96E+10
MDQ	5.34E+10	3.27E+10	2.22E+10	1.98E+10
MDQM	3.78E+10	2.69E+10	2.16E+10	1.99E+10
MDQR	5.26E+10	3.26E+10	2.22E+10	1.98E+10
SARR	5.74E+10	3.28E+10	2.22E+10	1.98E+10
NIR	5.37E+10	3.28E+10	2.22E+10	1.98E+10
NRR	9.15E+10	4.71E+10	2.49E+10	2.00E+10
MR	2.83E+10	1.73E+10	1.01E+10	2107101
SRBRR	5.52E+10	3.61E+10	2.33E+10	3893386

Table C.36: Maximum job stretch times (seconds) of scheduling algorithms for '5%' workload of AuverGrid

Scheduling Algorithms	Number of CPUs			
	16	**32**	**64**	**128**
FCFS	4.31E+07	17795355	4874222	2940306
LJF	56328718	27523277	7597035	4909032
SPN	9166762	4473588	3133244	4915792
P	4.63E+07	21696340	9276507	2081898
RR	52962.63	18422.53	4465.86	806.05
PLRR	4.30E+07	16923690	4665016	1103558
MH	446.16	221.52	156	252.72
MHM	422448.5	526401	159.5	769858.7
MHR	933	995.2	1107.16	933
MDQ	49767.84	16653.48	4208.31	735.09
MDQM	970040.1	873136.8	811511.7	560863.7
MDQR	169500.7	73078.75	17848.98	3782.33
SARR	27490611	503024.1	186325.8	22345.14
NIR	55017.6	18627.84	5491.2	982.08
NRR	4.36E+07	17171838	4483496	680789.6
MR	26506530	12321380	5942427	4234827
SRBRR	4.85E+07	24157375	12918321	7348587

Table C.37: Average waiting times (seconds) of scheduling algorithms for ‘10%’ workload of LCG1 by changing time quantum using ‘64’ CPUs

Scheduling Algorithms	**Time Quantum**			
	50	**1000**	**2000**	**5000**
RR	134544.8	138407.5	148557.7	199256.1
MH	59496.62	60317.14	61040.83	62997.25
MDQ	133811.1	129837.8	129299.2	131936.1

Table C.38: Average turnaround times (seconds) of scheduling algorithms for ‘10%’ workload of LCG1 by changing time quantum using ‘64’ CPUs

Scheduling Algorithms	Time Quantum			
	50	1000	2000	5000
RR	139992.7	143855.3	154005.6	204704
MH	64944.46	65764.98	66488.67	68445.09
MDQ	139258.9	135285.6	134747	137383.9

Table C.39: Average response times (seconds) of scheduling algorithms for ‘10%’ workload of LCG1 by changing time quantum using ‘64’ CPUs

Scheduling Algorithms	Time Quantum			
	50	1000	2000	5000
RR	1425.504	17615.06	48945.55	115516.4
MH	52955.72	53836.98	54642.57	56083.7
MDQ	2567.438	33279.72	49845.27	75747.99

Table C.40: Average slowdown times (seconds) of scheduling algorithms for ‘10%’ workload of LCG1 by changing time quantum using ‘64’ CPUs

Scheduling Algorithms	**Time Quantum**			
	50	**1000**	**2000**	**5000**
RR	58.80509	99.43363	229.5558	528.8822
MH	4.426133	6.121623	7.324399	9.112529
MDQ	58.87226	113.8643	157.1655	212.2047

Table C.41: Total completion times (seconds) of scheduling algorithms for ‘10%’ workload of LCG1 by changing time quantum using ‘64’ CPUs

Scheduling Algorithms	**Time Quantum**			
	50	**1000**	**2000**	**5000**
RR	3.79E+09	3.86E+09	4.05E+09	5.00E+09
MH	2.37E+09	2.39E+09	2.40E+09	2.44E+09
MDQ	3.77E+09	3.70E+09	3.69E+09	3.74E+09

Table C.42: Maximum job stretch times (seconds) of scheduling algorithms for ‘10%’ workload of LCG1 by changing time quantum using ‘64’ CPUs

Scheduling Algorithms	**Time Quantum**			
	50	**1000**	**2000**	**5000**
RR	1990	24208	47855	105850
MH	57	390	748	2748
MDQ	2922	35623	63000	32654