

STATUS OF THESIS

Title of thesis

COMPARATIVE PERFORMANCE OF SOME HYBRID HEURISTICS SCHEDULING ALGORITHMS BASED ON SLTR AND DEADLINE FOR COMPUTATIONAL GRID

I HARUNA AHMED ABBA

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:



Signature of Author

Dr. Mohamed Nordin Zakaria
Senior Lecturer
Endorsed by
Computer & Information Science Department
Universiti Teknologi PETRONAS
Bandar Seri Iskandar, 31300 Tronoh
Perak Darul Ridzuan, MALAYSIA

Signature of Supervisor

Permanent address: No.5, Phase 3,
Kundila Zaria Road, Kano State,
Nigeria

Name of Supervisor
Dr. Mohamed Nordin Zakaria

Date : 2/11/2013

Date : 2/11/2013

UNIVERSITI TEKNOLOGI PETRONAS
COMPARATIVE PERFORMANCE OF SOME HYBRID HEURISTICS
SCHEDULING ALGORITHMS BASED ON SLTR AND DEADLINE FOR
COMPUTATIONAL GRID

BY

HARUNA AHMED ABBA

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

Signature:

Dr. Mohamed Nordin Zakaria
Senior Lecturer
Computer & Information Science Department
Universiti Teknologi PETRONAS
Bandar Seri Iskandar, 32610 Seremban
Perak Darul Ridzuan, MALAYSIA

Main Supervisor:

Dr. Mohamed Nordin Zakaria

Signature:

A. Pal
21/1/2013

Co-Supervisor:

Dr. Anindya Jyoti Pal

Signature:

J. Jaafar
Dr. Jafreezal Bin Jaafar
Head
Department of Computer & Information Sciences
Universiti Teknologi PETRONAS
Bandar Seri Iskandar, 32610 Seremban
Perak Darul Ridzuan, MALAYSIA

Head of Department:

Dr. Jafreezal Bin Jaafar

Date:

10 3 JAN 2013

COMPARATIVE PERFORMANCE OF SOME HYBRID HEURISTICS
SCHEDULING ALGORITHMS BASED ON SLTR AND DEADLINE FOR
COMPUTATIONAL GRID

BY

HARUNA AHMED ABBA

A Thesis

Submitted to the Postgraduate Studies Programme

as a Requirement for the Degree of

MASTER OF SCIENCE

DEPARTMENT OF COMPUTER AND INFORMATION SCIENCES

UNIVERSITI TEKNOLOGI PETRONAS

BANDAR SERI ISKANDAR,

PERAK

JANUARY 2012

DECLARATION OF THESIS

Title of thesis

COMPARATIVE PERFORMANCE OF SOME HYBRID
HEURISTICS SCHEDULING ALGORITHMS BASED ON SLTR
AND DEADLINE FOR COMPUTATIONAL GRID

I

HARUNA AHMED ABBA

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.



Signature of Author

Witnessed by

Dr. Mohamed Nordin Zakaria
Senior Lecturer
Computer & Information Science Department
Universiti Teknikal
Semenanjung Malaysia
Johor Darul Ri'za, MALAYSIA

Signature of Supervisor

Permanent address: No.5, Phase 3,
Kundila Zaria Road, Kano State,
Nigeria

Name of Supervisor

Dr. Mohamed Nordin Zakaria

Date : 2/1/2013

Date : 2/1/2013

DEDICATION

To my beloved Father, Haruna Usman (May Allah reward you abundantly)

My beloved mother, my brothers and my sisters

ACKNOWLEDGEMENTS

In the name of Allah, the Most Gracious and the Most Merciful Alhamdulillah, all praises to Allah for the strength and His blessing in completing this thesis.

First of all, I would like to express my deepest sense of gratitude to my esteemed supervisor Dr. Mohamed Nordin Bin Zakaria and my co-supervisor Dr. Anindya Jyoti Pal, for their continuous support, patience, motivation, enthusiasm, and immense knowledge. I rode on the shoulders of their guidance in the course of the research and writing of this thesis. I do not think I could have better advisors and mentors for my Msc study.

My sincere thanks also goes to Dr. Syed Nasir Mehmood Shah for enlightening me the first glance of this research area.

I am grateful to the entire staff and faculty members of CIS department as well as my fellow research colleagues especially HPC research group members, for their support and love throughout my study. In particular, I would want to specially thank my colleagues Thayalan Sandran “Kya bate hai ji” and Djamalladine Mahamat Pierre for the stimulating discussions, support, motivation and for all the fun we had. Such shared moments helped to remove drudgery from my studies. I express my profound gratitude to Ugheoke Benjamin for his efforts in proof reading this thesis draft.

Finally, I seize this opportunity to express profound gratitude from the deepest parts of my heart to my beloved parents, grandparents, and my siblings for their love and continuous support both spiritually and materially, throughout my study period.

ABSTRACT

Grid computing can be described as form of distributed computing that involves collection of independent computers coordinating and sharing computing, application, data storage or network resources using high speed networks across dynamic and geographically distributed environment. Grid scheduling is an essential element of a Computational Grid infrastructure. Typical scheduling challenges tend to be NP-complete problems where there is no optimal solution. Research on Grid scheduling focuses in solving three challenges: finding a good algorithm, automating the process, and developing a scalable, robust and efficient scheduling mechanism. The complexities involved in scheduling challenges increase with the size of the Grid.

In different environment, users' priority is mainly focusing on job deadline. Even though deadlines are very important, no work has been done on scheduling algorithms combining real-time system and round robin fairness based on deadline, slack time to create fairness between the tasks and processors. Therefore, there is still a chance of improving it. This deadline and slack time can be derived from the operational research (OR) concept into grid scheduling. Some researchers have considered slack time using different techniques, but not yet considered operational research slack time concept by combining real-time system and round robin fairness techniques.

The research reported here therefore is focused on the development of grid scheduling algorithms based on deadline and slack time parameters, using the concept of operational research (OR). This is because, users main concern is to finish the jobs execution within the deadline upon his submission of jobs for execution.

The developed algorithms in this research were validated using real workload traces as benchmark on real grid computational environment. The results were compared with some baseline scheduling approaches in extant literature.

Based on the general observations, the research results have shown that the performances of grid scheduling algorithms developed and reported in this thesis give good results and also support true scalability, when in the scenario of increasing workload and number of processors on a real computational grid environment.

ABSTRAK

Pengkomputeran grid boleh digambarkan sebagai bentuk pengkomputeran teragih yang melibatkan koleksi komputer bebas menyelaraskan dan berkongsi sumber pengkomputeran, aplikasi, penyimpanan data atau rangkaian yang menggunakan rangkaian kelajuan tinggi di seluruh persekitaran yang dinamik dan geografi diedarkan. Penjadualan grid adalah satu elemen penting dalam infrastruktur Grid pengiraan. Cabaran penjadualan biasa cenderung untuk menjadi masalah NP-lengkap di mana tiada penyelesaian optimum. Penyelidikan mengenai penjadualan Grid memberi tumpuan dalam menyelesaikan tiga cabaran: mencari algoritma yang baik, mengautomatiskan proses, dan membangunkan penjadualan mekanisme berskala, mantap dan cekap. Kerumitan yang terlibat dalam penjadualan cabaran meningkat dengan saiz Grid.

Dalam persekitaran yang berbeza, keutamaan pengguna terutamanya memberi tumpuan pada tarikh akhir kerja. Walaupun tarikh akhir adalah sangat penting, tiada kerja telah dilakukan pada algoritma penjadualan menggabungkan masa sebenar sistem dan bulat robin keadilan berdasarkan tarikh akhir, masa kendur untuk mewujudkan keadilan antara tugas dan pemproses. Oleh itu, masih ada peluang memperbaiki ia. Kali ini tarikh akhir dan kendur boleh diperolehi dari penyelidikan operasi (OR) konsep ke penjadualan grid. Beberapa orang penyelidik telah dianggap masa kendur menggunakan teknik yang berbeza, tetapi tidak lagi dianggap penyelidikan kendur operasi konsep masa dengan menggabungkan masa sebenar sistem dan teknik keadilan pusingan robin.

Oleh itu, penyelidikan yang dilaporkan di sini memberi tumpuan kepada pembangunan algoritma penjadualan grid berdasarkan tarikh akhir dan parameter masa kendur, menggunakan konsep penyelidikan operasi (OR). Ini adalah kerana,

kebimbangan pengguna utama adalah untuk menamatkan pelaksanaan pekerjaan dalam tarikh akhir semasa penyerahan pekerjaan bagi pelaksanaan.

Algoritma yang dibangunkan dalam kajian ini telah disahkan menggunakan kesan beban kerja sebenar sebagai penanda aras terhadap alam sekitar grid pengiraan sebenar. Keputusan telah berbanding dengan beberapa pendekatan penjadualan asas dalam kesusasteraan wujud.

Berdasarkan pemerhatian umum, hasil penyelidikan telah menunjukkan bahawa prestasi algoritma penjadualan grid dibangunkan dan dilaporkan di dalam tesis ini memberikan hasil yang baik dan juga menyokong berskala benar, apabila dalam senario beban kerja yang semakin meningkat dan bilangan pemproses pada persekitaran grid pengiraan sebenar .

In compliance with the terms of the Copyright Act 1987 and the IP Policy of the university, the copyright of this thesis has been reassigned by the author to the legal entity of the university,

Institute of Technology PETRONAS Sdn Bhd.

Due acknowledgement shall always be made of the use of any material contained in, or derived from, this thesis.

© Haruna Ahmed Abba, 2012

Institute of Technology PETRONAS Sdn Bhd

All rights reserved.

TABLE OF CONTENT

CHAPTER 1 INTRODUCTION.....	1
1.1 Chapter Overview.....	1
1.2 Background of the Study.....	1
1.3 Grid Computing.....	2
1.4 Grid Scheduling.....	3
1.5 Scheduling Architecture.....	4
1.6 Research Motivation.....	5
1.7 Problem Statement.....	6
1.8 Research Objectives.....	7
1.9 Research Contribution.....	7
1.10 Outline of Thesis.....	8
CHAPTER 2 LITERATURE REVIEW.....	9
2.1 Chapter Overview.....	9
2.2 Previous Research on Grid Resource Allocation.....	9
2.2.1 Static Based.....	9
2.2.2 Dynamic Based.....	10
2.2.3 Linear Programming.....	10
2.2.4 Game Theory.....	10
2.2.5 Fuzzy Clustering.....	11
2.2.6 Market Based.....	11
2.2.7 Reinforcement Learning.....	12
2.3 Previous Research on Grid Scheduling.....	12
2.3.1 Evolutionary Algorithms in Grid Scheduling.....	12
2.3.2 Previous Research on Deadline and Slack Time Based Scheduling... 16	16
2.4 Summary.....	22
CHAPTER 3 METHODOLOGY.....	24
3.1 Chapter Overview.....	24
3.2 Research overview and Process Flow.....	24
3.3 Grid Scheduling Modeling.....	25

3.3.1 Benchmark Traces files.....	26
3.3.2 Resource Allocation.....	27
3.3.3 Scheduling Algorithms	28
3.3.4 Performance Metrics.....	29
3.3.5 Algorithm to Compute Find Parameters	29
3.4 Scheduling Algorithms.....	29
3.5 Experimental Procedures.....	53
3.5.1 Experimental Setup.....	53
3.5.2 Parameter Settings.....	53
3.5.3 Benchmark Description.....	54
3.6 Summary.....	56
CHAPTER 4 RESULT AND DISCUSSION.....	57
4.1 Chapter Overview.....	57
4.2 Comparative Performance Analysis of Scheduling Algorithms	57
4.2.1 Das-2 Traces.....	58
4.2.2 AuverGrid Traces.....	61
4.2.3 Grid5000 Traces.....	65
4.2.4 LCG Traces	69
4.2.5 NorduGrid Traces.....	73
4.2.6 Sharcnet Traces	77
4.2.7 Job Variations of Das-2 Traces	81
4.2.8 Job Variations Sharcnet Traces	91
CHAPTER 5 CONCLUSION AND FUTURE RESEARCH.....	104
5.1 Chapter Overview.....	104
5.2 Conclusion.....	104
5.2.1 Outcome of the Literature Review	104
5.2.2 Outcome of Algorithm Development	105
5.3 Research Limitation.....	106
5.4 Recommendations and Future Works	106
REFERENCES	118
APPENDIX A PUBLICATIONS.....	118

LIST OF FIGURES

Figure 1.1: Scheduling architecture.....	4
Figure 3.1: Research overview and process flow.....	24
Figure 3.2: Scheduling modeling.....	26
Figure 3.3: Master/Slave Architecture.....	27
Figure 4.1: Average Turnaround Time.....	58
Figure 4.2: Average Waiting Time.....	59
Figure 4.3: Maximum Tardiness.....	60
Figure 4.4: Average Turnaround Time.....	62
Figure 4.5: Average Waiting Time.....	63
Figure 4.6: Maximum Tardiness.....	64
Figure 4.7: Average Turnaround Time.....	66
Figure 4.8: Average Waiting Time.....	67
Figure 4.9: Maximum Tardiness.....	68
Figure 4.10: Average Turnaround Time.....	69
Figure 4.11: Average Waiting Time.....	71
Figure 4.12: Maximum Tardiness.....	72
Figure 4.13: Average Turnaround Time.....	73
Figure 4.14: Average Waiting Time.....	75
Figure 4.15: Maximum Tardiness.....	76
Figure 4.16: Average Turnaround Time.....	77
Figure 4.17: Average Waiting Time.....	79
Figure 4.18: Maximum Tardiness.....	80
Figure 4.19: Average Turnaround Time.....	81
Figure 4.20: Average Waiting Time.....	83
Figure 4.21: Maximum Tardiness.....	85
Figure 4.22: Average Turnaround Time.....	86
Figure 4.23: Average Waiting Time.....	88
Figure 4.24: Maximum Tardiness.....	90
Figure 4.25: Average Turnaround Time.....	92

Figure 4.26: Average Waiting Time.....	94
Figure 4.27: Maximum Tardiness.....	96
Figure 4.28: Average Turnaround Time.....	98
Figure 4.29: Average Waiting Time.....	100
Figure 4.30: Maximum Tardiness.....	102

LIST OF TABLES

Table 4.1: Standard Deviation	59
Table 4.2: Standard Deviation	60
Table 4.3: Standard Deviation	61
Table 4.4: Standard Deviation	63
Table 4.5: Standard Deviation	64
Table 4.6: Standard Deviation	65
Table 4.7: Standard Deviation	66
Table 4.8: Standard Deviation	67
Table 4.9: Standard Deviation	69
Table 4.10: Standard Deviation	70
Table 4.11: Standard Deviation	71
Table 4.12: Standard Deviation	73
Table 4.13: Standard Deviation	74
Table 4.14: Standard Deviation	75
Table 4.15: Standard Deviation	77
Table 4.16: Standard Deviation	78
Table 4.17: Standard Deviation	80
Table 4.18: Standard Deviation	81
Table 4.19: Standard Deviation	82
Table 4.20: Standard Deviation	84
Table 4.21: Standard Deviation	86
Table 4.22: Standard Deviation	88
Table 4.23: Standard Deviation	90
Table 4.24: Standard Deviation	91
Table 4.25: Standard Deviation	93
Table 4.26: Standard Deviation	95
Table 4.27: Standard Deviation	97
Table 4.28: Standard Deviation	100
Table 4.29: Standard Deviation	102

Table 4.30: Standard Deviation 103

CHAPTER 1

INTRODUCTION

1.1 Chapter Overview

This chapter lays the foundation for the entire research. It begins by giving background information to the problem in section 1.2, followed by grid computing in section 1.3, grid scheduling in section 1.4, scheduling architecture in section 1.5, and research motivation in section 1.6, problem statement in section 1.7, research objectives in section 1.8, research contribution in section 1.9 and ends with outline of thesis in section 1.10.

1.2 Background of the Study

In recent years, increasing demand for computing [1-5] has led to the development of computational grid. A computing grid uses the idle time of thousands or millions of computers throughout the world [6]. Precisely, a grid is a large-scale, heterogeneous, dynamic collection of independent computers, geographically distributed and interconnected with high speed networks. The resources making up a grid need to be managed to provide a good quality service. Resource allocation is one of the major problems in grid due to large-scale heterogeneity both in processing speed and interconnection speed between different computers. In fact, resource allocation is also an NP (non-deterministic polynomial-time) complete problem [7] where there is no optimal solution. This new field has emerged, distinguished from traditional distributed computing by its concentration on large-scale resource sharing. Also it's been shown that resource heterogeneity affects the resource allocation in a significant way, in terms of performance, reliability, robustness and scalability.

To facilitate job scheduling as well as resource management in grid, a resource scheduler or a meta-scheduler has to be used. A scheduler is essential in any multi-user grid environment. The task of the grid resource scheduler is to identify dynamically, characterize the accessible resources and select the right resource for jobs submission. While much has been done on grid scheduling, due to its NP-complete nature, grid scheduling problem continues to be analyzed broadly in different areas.

1.3 Grid Computing

The word “Grid” a word borrowed, from the electric energy power grid, derives from significantly pervasive, easily available resource that originates at distributed sites, each with potentially different system and conditions. Grid computing originated in the early 1990’s. Ian Promote[8] was promoting a program to elevate shared computing to some global level. Just like the internet which is a tool for mass communication, grids are the tools that make computer resources and space globally available for storage. Grid computing is all about getting computer systems to operate together. In fact according to Zhang *et al.*, [9], grid assist to promote the web to some true computing platform, mixing the characteristics and services information about enterprise computing having the ability to share heterogeneous distributed assets-from programs, data, storage and servers. Moreover, a definition given by the Globus Alliance (an investigation and development initiative focused on enabling the application of grid concepts to scientific and engineering computing) [10], is as follows:

“The grid describes an infrastructure that enables the integrated, collaborative usage of high-finish personal computers, systems, database, and scientific instruments possessed and handled by multiple organizations.”

There are three major areas that have drawn much attention from grid researchers and developers. Prime concern is resource sharing. Resource sharing may be the prime cause of the grid philosophy [1]. Grid is all about putting systems in position to ensure that everybody benefits from the efficiencies of sharing. Moreover, grid gives use of extra computing power and may compute things that cannot be computed using

only one computer. The second large idea behind grid is safe access which can be a direct result of the very first large idea. However, to ensure secure access, grid developers and users have to manage the following three essential things[11]:

- Access policy
- Authentication
- Authorization

The third large idea behind grid is efficient utilization of resources, because it includes a mechanism to allocate jobs effectively and instantly among many users, and as a result may lessen waiting time.

1.4 Grid Scheduling

Grid scheduling is understood to be the entire process of making scheduling choices, relevant resources over multiple administrative domain names. This method may include searching multiple administrative domain names to utilize a single machine or scheduling just one job to make use of multiple resources in a single site or multiple sites. Scheduling is a process that maps and handles execution of interdependent tasks on distributed resources. It introduces allocating appropriate assets to workflow tasks to ensure that the execution could be implemented to satisfy objective functions per customers. Scheduling has two important definitions. First of all, scheduling is a decision making function; to determine a schedule. Secondly, scheduling is a body associated with a theory; it is actually a collection of principles, models and methods. Proper scheduling has a significant effect on the performance of the system. Generally, the issue of mapping tasks on distributed resources goes to some class of problems referred to as NP-complete problems [12]; for such problems, no known algorithms can create the optimal solution within polynomial time. Scheduling function can be the actual allocation of resources over time in order to perform a collection of task raised in a variety of scenarios.

Casavant *et al.* [13], categorized task scheduling in distributed computing software as ‘local’ task scheduling and ‘global’ task scheduling. Local scheduling involves handling a job of tasks to time-slices of merely one resource whereas global

scheduling involves determining where to carry out a task. Based on this definition, scheduling is a type of global task because it concentrates on mapping and controlling the execution of interdependent tasks on shared resources that are not directly under its control. However, the scheduler must coordinate with diverse local management systems as grid resources are heterogeneous when it comes to local configuration and guidelines. Users' QoS (quality of service) constraints can also be essential in the scheduling process in order to satisfy their needs.

1.5 Scheduling Architecture

The architecture of the scheduling infrastructure is essential for scalability, autonomy, quality and performance from the system[14]. There are three major groups of workflow scheduling architecture, which are shown in Figure 1. They are centralized, hierarchical and decentralized scheduling schemes.

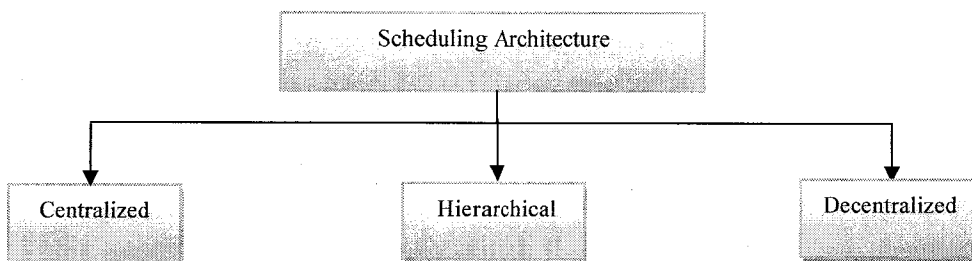


Figure 1.1: Scheduling architecture.

In a centralized workflow enactment environment, one central workflow scheduler makes scheduling choices for all tasks within the workflow. Moreover according to T. A. A. Project[14], the scheduler has the data concerning the entire workflow and collects information of available processing resources and produce efficient scheduling, since it has all necessary information. However, it is not scalable regarding the amount of tasks, the classes and quantity of grid resources. It is thus only appropriate for any small scale workflow or perhaps a massive workflow by which every task has got the same objective (e.g. same class of resources). Unlike centralized scheduling, both hierarchical and decentralized scheduling enables tasks to become scheduled by multiple schedulers. Therefore, one scheduler only keeps the

data associated with a sub-workflow. Thus, in comparison to centralized scheduling, they're more scalable given that they limit the amount of tasks handled by one scheduler. However, the very best decision designed for an incomplete workflow can lead to sub-optimal performance for that overall workflow execution. Furthermore, conflict troubles are more serious [15]. An example of conflict is the fact that tasks from different sub-workflows scheduled by different schedulers may compete for similar resource. However, for hierarchical scheduling, there is a central manager and multiple lower level sub-workflow schedulers. This central manager accounts for manipulating the workflow execution and setting the sub-workflows from the low-level schedulers. For instance, in Gridflow project [16], there is one workflow manager and multiple lower level schedulers. The workflow manager plans sub-workflows onto corresponding lower level schedulers. Each lower level scheduler accounts for scheduling tasks inside a sub-workflow onto assets possessed by one organization. The main benefit of using hierarchical architecture would be the different scheduling guidelines which could be used within the central manager when minimizing the level of schedulers [14]. The failure of the central manager can lead to entire system failure.

1.6 Research Motivation

Grid scheduling is one of the major challenges of grid computing, where use of scheduling techniques is frequently required. Grid scheduling challenge is generally based on some resources (typically machines, storage, memory, network, etc.). The number of submitted jobs in grid environment is typically large. These jobs are ordered in a queue, and scheduling approaches are used to order the jobs as well as delivering them to the right destination resources.

There are a couple of reasons why grid scheduling is such challenging.

- Firstly, increase in number of jobs increases the demands of the search space.
- Secondly, the factors that determine optimal scheduling are dynamic in nature.
- Thirdly, domains and applications need solutions of different variations associated with the scheduling problem.

Moreover, no work has been done on scheduling algorithms combining real-time system and round robin fairness based on deadline, slack time to create fairness between the tasks and processors. Why fairness? Operating system and some distributed system environment (BOINC) were integrated with round robin techniques in order to create fairness between the tasks as well as processors to avoid idle state.

In other words, real-time systems, as well as their deadlines, are classified by the consequence of missing a deadline. Firstly, a hard real-time system ensures that when an event occurs, it should be executed within its deadline time at all times in a given hard real-time system. Instead, in soft real-time systems the precedence and sequence of task operations are defined, interrupt latencies as well as context switching latencies are small, but there can be few deviations between expected latencies of the tasks and observed time constraints. Nevertheless, few missed of deadline are accepted.

Therefore, this research aimed at the development of soft real-time grid scheduling algorithms based on deadline and slack time parameters inherited from operational research (OR) on software project management concept in to grid scheduling.

1.7 Problem Statement

Grid scheduling is one of the major challenges of a grid environment, due to large scale heterogeneity. In different environment, users' priority is mainly on job deadline. Deadline is very important. But not much has been done on scheduling algorithms based on deadline and slack time, especially in distributed grid environment.

Therefore, there is still a chance of improvement based on deadline and slack time, using techniques and concepts of operational research (OR) in solving grid scheduling problems. Some researchers [17, 18] have considered slack time using different techniques, but not yet considered operational research slack time concept by combining real-time system and round robin fairness techniques.

The research reported here therefore is focused on the development of grid scheduling algorithms based on deadline and slack time parameters, using the concept of operational research (OR) [19]. This is because, users main concern is to finish the jobs execution within the deadline upon his submission of jobs for execution.

In addition, most of the existing scheduling algorithms available in literature were not developed and benchmarked using real workload traces. This may impair their efficiency and robustness during application in real computational grid environment with respect to the following performance metrics:

- Average turnaround time
- Average waiting time
- Maximum tardiness

This forms the crux of the present research, which also calculated the standard deviations of the above parameters.

1.8 Research Objectives

The objectives of this research are as follows:

- To integrate new robust hybrid methods based on baseline approaches.
- To implement, evaluate and test these developed algorithms with real benchmark traces on real computational grid environment.

The outcome of the research work will lead to a significant improvement in the efficiency and robustness in Grid scheduling.

1.9 Research Contribution

This research gives better insights and idea or solution for scheduling technique through deadline versus time for multiple jobs on a limited resource. The main contributions of this research are as follows:

- Integrated new robust hybrid methods based on baseline approaches.

Implemented, evaluated and tested these developed algorithms with real benchmark traces on real computational grid environment.

1.10 Outline of Thesis

The outline for this thesis is as follows:

- Chapter 2 presents a discussion on relevant literature on related research, thereby putting Grid scheduling and the processes of scheduling in the right perspective. This is followed by a brief explanation of scheduling techniques.
- Chapter 3 discusses the methodology used in this project. It covers experimental design as well as performance indices for evaluation.
- Chapter 4 highlights the results of the experiments conducted. There is also a discussion, which includes analysis and result comparison of the performance evaluation done.
- Chapter 5 is the conclusion and the future scope of this work.

CHAPTER 2

LITERATURE REVIEW

2.1 Chapter Overview

This chapter reviews the concepts of grid computing, scheduling, jobs distribution as well as the previous approaches employed in handling grid scheduling problems. The literature review provides much broad, but deep insight into extant approaches and reasons informing the selection of the technique used in this research to proffer robust, efficient, effective and accurate scheduling of grid problems.

2.2 Previous Research on Grid Resource Allocation

A brief overview of some previous researches based on different type of approaches that have been and are still being used in grid resource allocation are as follows:

2.2.1 Static Based

A static based resource allocation constitutes a fixed data entry or fixed accounting scheme such as a fixed access to a computer node. Tibor approach [20], main objective is to assign an application process to compute servers that can present the required Quality of Service as well as execute the processes in a cost-efficient manner. In a related development by Somasundaram *et al* [21], incoming jobs from different users are collected and stored in a job list and available resources are stored in a resource list. Swift scheduler (SS) in GridSim [22] maps jobs from resource queue as well as resources from job queue by the use of heuristic function. In Swift

Scheduler, job allocations as well as resource selection process are executed using a Shortest Job First policy, which minimizes the average waiting time for jobs.

2.2.2 Dynamic Based

A dynamic based resource allocation is a process whereby dynamic mechanisms adapt allocation according to the change of available resource quantities. The method has combines best fit algorithm and process migration [23]. Using to this approach, a resource reservation is decided by an administrator based on monitoring outcome specified by the system at a given time. Moreover, a global grid network [4] is presumed, where resources are distributed all over the globe. Users put forward applications to their local area network scheduler. However, the scheduler select resources related to the application requirements and allocate them to the requesting application. In order to achieve the resource virtualization, adapter design pattern is used [24].

2.2.3 Linear Programming

Linear programming is the process of taking various linear inequalities relating to some situation and finding the “most effective” value obtained under those conditions. Jun *et al.* [25], highlighted on the solution of resource allocation problems in a gigantic wireless network by applying linear programming. Specifically, they look into link scheduling problem assigning each link a collection of time slots in which it will transmit. The schedule sought is the one that can guarantee all links in each slot which it can transmit at the same time without triggering unexpected mutual interference.

2.2.4 Game Theory

Game theory is concerned with decision making in situation whereby two or more rational opponents are competition with conflicting interests in expectation of definite

outcomes in a given period of time [26]. The idea of applying game theory and economics to resource management has been covered in many work [27-32]. The majority of them investigated the economy in general equilibrium. In a research work by Weng *et al.* [32], game theory is utilized to optimize resource allocation. This approach takes advantage of proportional resource sharing model to manage grid resources. Based on this model, it is shown that the percentage of resource allocated to the user application is proportional to the bid value in comparison to other users' bids.

2.2.5 Fuzzy Clustering

Fuzzy clustering is a type of algorithm for cluster analysis whereby the allocation of data being positioned to clusters is not "hard" all-or- nothing. The algorithm by FuFang *et al.* [33], delicately assigns appropriate resource while reserving the resources whose power greatly exceed the requirements of current tasks for future use when complex large-scale tasks arrive to the very task that exactly suit its needs for resource. According to the work of Ru-Huai [34], the fuzzy clustering result can be obtained directly from fuzzy similarity matrix by using net-mask method. Through the use of net-mask method, the efficiency of this algorithm has been largely enhanced. Dawei *et al.* [35], proposed a novel heuristic grid resource allocation algorithm based on cluster grid resources.

2.2.6 Market Based

Market based resource allocation comprises of auctioneer who acts as a mediator between sellers and buyers' orders (requests). In this model, sell orders (offers) may be submitted at any time during the trading period. Within this frame work, Ferguson *et al.* [36], deal with utility functions, used for calculating the utility of resource allocation through the use of the utility based optimization method which will permit the integration of different optimization objectives into allocation process. Li *et al.* [37] and R. Buyya *et al.* [38], proposed a distributed computational economy-based framework, called as the Grid Architecture for Computational Economy

(GRACE), for resource allocation that regulate supply and demand of the available resources.

2.2.7 Reinforcement Learning

Reinforcement Learning is a type of Machine Learning and also a branch of Artificial Intelligence which enables machines and software agents to automatically determine the ideal behaviour within a specific context, in order to maximize its performance. Based on this method, Adil *et al.* [39], used the method of reinforcement learning by allowing a system which consists of a large number of heterogeneous reinforcements learning agents that share common resources for their computational needs. Mateescu *et al.* [16], created a simplified multi agent model of resource allocation based on the same concept.

2.3 Previous Research on Grid Scheduling

A brief overview of some previous researches based on different types of grid scheduling approaches is given in this section. In recent years, many researchers have offered different types of methods as well as different types of algorithms for dynamic job scheduling in different notion.

2.3.1 Evolutionary Algorithms in Grid Scheduling

Cao *et al.* [40], used Fuzzy C-Mean and Genetic Algorithms for dynamic job scheduling. This model introduces cluster analysis for classification of job characteristics (or objects), according to similarities among them, and for organizing objects into groups. Cluster is a group of objects that are more similar to each other than to objects in other clusters. Similarity is often defined by means of distance based upon the length from a data vector to some prototypical object of the cluster. The data are typically observations of some phenomenon. Each object consists of measured variables, grouped into a dimensional column vector as well as mapping the

jobs to the appropriate resources primarily based on Genetic algorithm. Furthermore, this approach separates workload data to three classifications based on jobs run-time historical data. In related work by Siriluck *et al.* [41], a static job scheduling algorithm through the use of the Fuzzy C-Mean along with Genetic algorithms appears to have been applied. This approach presents the strategies (ways) of allocating jobs to distinct nodes, which have been developed for predicting the characteristics of jobs (using Genetic Algorithms (GA) techniques) running in the grid environment. The researchers (Siriluck *et al.* [42]), presented the results of the simulation of the grid environment with regard to job allocations to distinct nodes. However, fuzzy clustering methods allow for uncertainty in the cluster assignments. Rather than partitioning the data into a collection of distinct sets (where each data point is assigned to exactly one set), fuzzy clustering creates a fuzzy pseudo partition, which consists of a collection of fuzzy sets. Fuzzy sets differ from traditional sets in that membership in the set is allowed to be uncertain. The results prove the model by using Fuzzy c-mean clustering approach for predicting the characterization of jobs as well as optimization involving jobs scheduling in grid environment. This kind of prediction and optimization engine provided jobs scheduling based upon historical information.

In another study (Florin Pop *et al.* [43]), a fault-tolerant scheduling framework through DIOGENES ("Distributed optimal genetic algorithm with respect to grid application scheduling"), was presented. This framework maps the actual architecture of MedioGRID, which is a real-time satellite image processing system operating within a Grid environment. The proposed solution provides a fault tolerant mechanism of mapping the image processing applications, on the available resources in MedioGRID clusters and uniform access.

Kamalapur *et al.* [44], presented an evaluation of recommended GA based scheduling against existing traditional algorithms. Individual solutions are randomly generated to form an initial population. Successive generations of reproduction and crossover produce increasing numbers of individuals in solution regions. The algorithm favors the fittest individuals. However, to achieve minimum waiting time the fitness function defined here is based on Shortest Job First algorithm. Fitness

function checks the jobs, which occurs after the crossover point. If the jobs present after crossover point has minimum CPU burst then fitness function marks it as fit to generate new generation. The process is repeated till a termination criterion is reached and termination criterion is a minimum waiting time.

In the work of Bouyer *et al.* [45], a meta-heuristic algorithm based on genetic algorithm to solve the workflow scheduling problem with the objective of minimizing the time and cost of the execution was presented. The newly introduced method consists of two phases: Phase 1: primary processing of information to make Decision Tree and assigning each existing record to its related class (Decision Tree of course, should be made and preserved only once). Phase 2: final processing and predicting the situation of a record, by using neighbouring records. In the first phase, to identify the main and efficient parameters in the existing database system and to clarify their effect on the final result, a processing operation is performed on it. The second step involves an attempt to classify the information into different classes (by using a decision tree classifier). At second phase, the desired record is first inserted in its class according to previously done classification in Decision Tree. Then considering the number desired of neighbours, the existing records are selected, which are similar to the desired record, upon which the predicting operation is then perform.

In the work of Ivan *et al.* [46], new job scheduling policy was determined by backfilling (JR-backfilling). The main goals of these policies were to decrease the workload execution time frame, a job waiting time, job response time, and average bounded slowdown and to successfully optimize the resource utilization.

Another method known as the particle swarm optimization (PSO) algorithm has been studied by Bu Yan-Ping *et al.* [47]. It uses discrete coding rule for grid scheduling with regard to the optimization of grid task scheduling problems and it optimizes the grid resources allocation. In the grid environment, the scheduling problem is to schedule a stream of tasks to a set of nodes. During the execution, there are some communications between nodes. The function of DPSO is to find the best tasks scheduling strategy and to obtain the optimal makespan. PSO is a population based stochastic optimization technique. It is first initialized with a group of random particles (solutions). In every iteration, each particle is updated by following two

“best” values, i.e., the personal best (pbest) and the global best (gbest). Pbest is the best solution (fitness) one particle has achieved so far. While gbest is the best value obtained so far by any particle in the population. Similarly, Mathiyalagan *et al.* [48], implemented a new approach based on particle swarm optimization algorithm in order to resolve task scheduling challenges in grid. The newly developed algorithm is generating an optimal schedule to complete task process within a minimum time frame as well as utilizing the resources in an efficient way. The performance of each particle is measured using a fitness function that varies depending on the optimization problem. Each particle in the swarm is represented by the following characteristics: the current position of the particle and the current velocity of the particle. Moreover, the particle swarm optimization which is one of the latest evolutionary optimization techniques conducts searches using a population of particles. Each particle corresponds to individual in evolutionary algorithms. Each particle has an updating position vector and updating velocity vector by moving through the problem space. In related work, Pooranian *et al.* [49], proposed a novel approach based on hybrid PSO and GELS (GPSO) algorithm in order to resolve grid scheduling challenge in order to attenuate makespan as well as missed task.

The approach introduced by Raksha Sharma *et al.* [50] reduces processing time frame and utilizes grid resource adequately. The primary goal is to maximize the resource utilization and reduce the processing time frame of jobs. However, the grid resource selection approach is based on Max Heap Tree (MHT) which best suits many large scale applications and the root node of MHT is selected for job submission. Somasundaram *et al.* [21], developed incoming jobs from different users that are collected and stored in the job list and available resources, which are stored in resource list, using the method of swift scheduler. The swift scheduler allocates jobs, and selects resource using a heuristic searching algorithm based on Shortest Job First (SJF), which minimizes the average time jobs spend on queues. Therefore, in general the turnaround time is minimized and resource utilization is optimized.

In the work of Asgarali *et al.* [51], a new approach on fault tolerance mechanisms for the resource scheduling on the grid was proposed by applying a method called Rough Set Analysis algorithm on grid nodes (provider nodes) and optimized case-

based Reasoning (OCBR) algorithm on scheduler machine, for prediction, detection and recovery of faults in grid. OCBR is one of the preferred problem-solving strategies and machine learning techniques in complex and dynamically changing situations. However, the proposed grid-scheduling approach can select the best fault tolerance nodes and also detect a failed node and simply manage it by using one of the provided strategies such as multi-versioning, reservation queue and replacement, and transferring jobs to the nearest neighbour. OCBR-executer classifies the existing Rules (received from all nodes) based on similarity to the new desired job. At first, the information or primary system parameters are identified and integrated. Next, the final result of the problem is obtained by performing the final processing among the desired record and its neighbours (in the same class).

2.3.2 Previous Research on Deadline and Slack Time Based Scheduling

In the work of Miyagi *et al.* [52], a deadline aware scheduling scheme for the lambda grid system was proposed to support a huge computer grid system based on an advanced photonic network technology. In the lamda scheme, the assignment of wavelengths to jobs in order to efficiently carry various services that is very critical in grid networks. Such services have different requirements such as the job completion deadlines, and wavelength assignment must consider the job deadlines. However, the proposed scheme uses deadline first as priority and then assigns time slots to a call over time according to its deadline, which allows it to increase the system performance in handling short deadline calls.

In the work of Rajkumar Buyya *et al.* [53], deadline and budget constraint (DBC) which allows allocation of resources depending on the user's QoS requirements, such as the deadline, budget, and optimization strategy were proposed. The proposed algorithm called cost time optimization was developed and evaluated using the GridSim toolkit by comparing its performance and a lot of service delivery with the cost optimization. When there are multiple resources with the same cost and capability, the cost time optimization algorithm schedules jobs on them using the time optimization strategy for the deadline period.

Lui *et al.* [54], constructed a novel Generalized Distributed Scheduler (GDS) for tasks with different priorities and deadlines. They considered a non-pre-emptive scheduling strategy applied over a bag of independent mixed tasks in computational grids. Tasks are ranked based upon priority and deadline. Tasks are shuffled to earlier points to pack the schedule and create fault tolerance. However, dispatching is based upon task-resource matching and accounts for computation as well as communication capacities.

The work of Eddy *et al.* [55], was an extension of the work in Takefusa *et al.* [56]. They considered both priority and deadline of the tasks to select a server. They showed that a good number of tasks can meet their deadlines by the increase of 1) using task priorities and 2) using a fallback mechanism to reschedule tasks that were not able to meet their deadline on the selected servers.

Alexis *et al.* [57], presented a policy that they called Repeated Placing Policy. The policy did not ignore jobs and considers a job's execution within a short period of time before the global deadline. The drawback perceived in their work however, is that considering the jobs too early may cause many jobs to fail.

In [58] a resource characteristic based optimization method (RCBO), was combined with Earlier Gap, Earliest Deadline First (EG-EDF) policy to schedule jobs in a dynamic environment based on [59]. In the application of RCBO, each time new jobs arrival reaches a value of ten, RCBO is applied to change the positions of some jobs that have already arrived and are waiting in the schedules of some machines. RCBO may move the jobs to a better evaluated position.

Li-Ya *et al.* [60], studied and simulated Min-minII and Min-min heuristic as the benchmark of the scheduling problems in a dynamic grid computing environment. However, based on the study, the Min-minII dynamic scheduling heuristic was used in order to utilize task deadline and task assignment time to testify that it can outperform Min-min makespan. In contrast, Shupeng Wang *et al.* [61], have proposed a Survivability-Based scheduling algorithm for bag-of-tasks applications with Deadline Constraints (SBDC), that maximizes the survivability while meeting the deadline for delivering results. An algorithm which integrates the ideas of a classical

bin packing (Best Fit) and a mixed integer quadratic programming modelling approach has been used by Cong Liu *et al.* [62]. This approach, which is known as Residual Capacity Maximization Scheduling (RCMS), is highly scalable as it does not need to know the global state of the grid. RCMS prioritizes tasks according to task types as well as the deadline. Moreover, RCMS proposes a mixed integer quadratic programming model that always maximizes the residual capacity of resources at each step following a task-resource mapping.

In the research of Spooner *et al.* [63], the problem of grid workload management has been resolved through the development of a multi-tiered scheduling architecture (TITAN) that employs a performance prediction system (PACE) and task distribution brokers to meet user-defined deadlines and improve resource usage efficiency. PACE is used to obtain parallel application performance data prior to run-time allowing resource requirements to be expected and deadlines considered.

In the work of Fang Dong *et al.* [64], a Grid tasks scheduling strategy based on QoS priority grouping is proposed. In this algorithm, the deadline property of task, acceptance rate of tasks and makespan of systems is comprehensively considered. Moreover, scheduling is based on task priority grouping and deadline.

Hiroyuki *et al.* [65], proposed a deadline-scheduling scheme for wavelength assignment in grid networks that can meet QoS (Quality of Service). Moreover, Hiroyuki *et al.* [65] came up with a combination of two ideas: Deadline-first reservation and Greedy tentative reservation; in order to improve the utilization of current time slots at longer wavelengths after successfully reserving the slots. The proposed approach assigns time slots to a call over an extended period according to its deadline. This makes more time slots available for short deadline calls. If no short deadline calls are received, the time-slots reserved for them are wasted.

The work of Vasumathi *et al.* [66], shows that combining redundant scheduling with deadline-based scheduling could lead to a fundamental tradeoff between throughput and fairness. Vasumathi *et al.* [66] came up with a new scheduling algorithm called Limited Resource Earliest Deadline (LRED) that couples redundant

scheduling with deadline driven scheduling in a flexible way by using a simple tunable parameter to exploit this tradeoff.

In another related work Saurabh *et al.* [67], presented MaxCTT and MinCTT. They also presented a cost metric to manage the trade-off between the execution cost and time. In their work, various user requirements were considered during scheduling simultaneously in terms of cost and execution time. Since concurrent users may generate conflicting schedules to access the same resources which are cheaper and faster.

Dalibor *et al.* [68], applied a technique known as Earliest Gap- Earliest Deadline First (EG-EDF). This technique fills earlier existing gaps in the schedule with newly arriving jobs. If no gap for a coming job is available EG-EDF rule uses Earliest Deadline First (EDF) strategy for including new job into the existing schedule. Scheduling choices are taken to meet the Quality of Service (QoS) requested by the submitted jobs, and to optimize the usage of hardware resources.

Kamalam *et al.* [69] applied a Divisible Load Theory (DLT) and Least Cost Method (LCM) to model the grid scheduling problem involving multiple worker nodes in each cluster. They came up with a hybrid job scheduling algorithm that minimizes the overall processing cost of the job and divisible job scheduling algorithm that minimizes the overall processing time of the job in a grid system that may consist of heterogeneous hosts.

Hongwei *et al.* [70], proposed a algorithm that uses a DAG (Directed Acyclic Graph) to locate the critical path, acquire the deadline of each task to compute their PRI (priority). The algorithm takes the below problems into consideration: the request of the user, the type of resources and re-scheduling of failed tasks. To meet users' requirements, Hongwei *et al.* [70] introduced the notion of users' urgent degree.

Ba Wei *et al.* [17], used the linguistic fuzzy sets to describe the period and the slack time of tasks that have uncertain characters. The threshold coefficient gotten by fuzzy rules assigns the threshold of the running task dynamically. Tasks are ordered by their slack time as in Least Slack First (LSF), however, some differences are made

that the threshold of the running task is got and it is considered as its slack time until the task is released or finished. Therefore, two characters are considered to judge the priority of a task, by its slack time and its threshold.

Bo Li *et al.* [71], presents a mechanism that can comply with the requirements of current advance reservation (AR) job, but at the cost of inflexibility of accepting and scheduling of other AR and non-AR jobs. In order to reduce the impact of a current reservation on other jobs, current AR job can start within a time span, ranging from its required start time with an additional slack time, instead of the rigid start time. The difference between its ready time with its latest start time can be defined as the job's slack time to begin to run.

In the work of Hwang *et al* [18], maximum occupation time (MOT) was defined and it limits the maximum time that one job (or a task) can have occupying one for processor. The MOT is determined by the timing constraints of a given task set. The aim of Least Slack Time Rate first (LSTR) is to ensure that no idle state is allowed in any processor. All tasks have the deadline and execution time as a timing constraint. The scheduler determines the task at the scheduled time to be executed on a processor. Tasks are executed on the processor(s) and then both the remaining execution time and the remaining deadline of these tasks decrease.

The work of Behera *et al.* [72], presented an improved Least-Laxity-First Algorithm (ILLF). Using a LLF scheduling algorithm, if two or more tasks have same laxities, laxity-tie occurs. Once laxity-tie occurs, context switches takes place at every scheduling point until the tie breaks. The laxity-tie in the LLF scheduling algorithm results in poorer system performance due to the frequent context switches. The improved Least Laxity First Scheduling Algorithm with intelligence time slice finds the time quantum by taking the greatest common divisor (GCD) of all the execution time of the processes. After every unit of time slice the laxity of each remaining process (present in the ready queue) is calculated. The loop continues until all the processes are being executed by the CPU.

In Golnar *et al.* [73], a distributed scheduler of workflows with deadlines in a P2P computing platform has been presented. It is a completely decentralized model, which

has been a validated using simulation that has shown good response times and low overhead in a system with one million nodes. Big workflows with highly concurrent tasks can be easily scheduled with low overhead and a good speedup.

Javier *et al.* [74], presented a distributed algorithm referred to as Resource-Aware Dynamic Incremental Scheduling (RADIS) strategy. The strategies were purposely designed to handle large volumes of computationally intensive arbitrarily divisible loads submitted for processing at cluster or grid systems involving multiple sources and sinks (processing nodes). In the same vein, Sivakumar *et al.* [75], considered a real-life scenario, wherein the buffer space (memory) available at the sinks (required for holding and processing the loads) varies over time, and the loads have deadlines and propose efficient “pull-based” scheduling strategies with an admission control policy that ensures that the admitted loads are processed, satisfying their deadline requirements.

Nikolaos *et al.* [76], proposed a new algorithm for fair scheduling. This algorithm uses a Max-Min fair sharing approach for providing fair access to users. When there is no shortage of resources, it assigns to each task enough computational power for it to finish within its deadline. When there is congestion, the main idea is to fairly reduce the CPU rates assigned to the tasks so that the share of resources that each user gets is proportional to the user’s weight. The weight of a user may be defined as the user’s contribution to the infrastructure or the price he is willing to pay for services or any other socioeconomic consideration.

In the work of Jia Yu *et al.* [77], a cost-based workflow scheduling algorithm was presented in order to minimize the cost of execution while reaching the deadline. A Markov Decision Process approach was utilized in order to schedule in a stepwise manner workflow task execution, such that it could possibly find the optimal path among services to execute tasks as well as transfer input or output data. However, to be more efficient, some additional priorities need to be considered, like maximum turnaround time and time delayed when it comes to the rescheduling of unexecuted job.

Sulistio *et al.* [78] built a resource set of jobs which minimize cost or time, depending upon the user's preferences as well as deadline and budget constraints. The algorithm minimizes either the overall cost or the time of execution depending on the user's preference, subject to two user deadline constraints: the deadline by which the processing must be completed and the overall budget for performing the computation. In doing so, it is guided by factors such as cost and speed of accessing, transferring and processing data. Each job requires one or more datasets as input. Each dataset is available through one or more data hosts. Moreover, the scheduler gathers information about the available compute resources and about the datasets and data hosts. It then makes a decision on where to submit the job. The job is dispatched to the selected remote computer resource where it requests for the datasets from the replica locations selected by the scheduler. After the job has finished processing, the results are sent back to the scheduler host or other storage resource.

The work done by Daphne *et al.* [79] aimed at dealing with the fairness problem by dropping the service time frame error. Their technique assigns to each task sufficient computational power to complete it within its deadline. The resources that each user gets are proportional to the user's weight or perhaps a shared. Here, scheduling of tasks is based on an error, called the service time error, that promotes fairness among users. However, it will be more optimized if priority is given based on the minimum time of execution of job, not on individual demand.

2.4 Summary

This chapter presented the literature review based on previous works relating to scheduling in grid environment. The importance of the review is underscored by its provision of overall points of interest and detailed explanation on particular topics relating to the planned research. It pinpoints the issues pivotal to grid scheduling that must be integrated within the applicable models. Though the review has shown that lots of research on grid scheduling has been reported, it is clear that much of these reported algorithms were developed with a motivation different from that in this

thesis. This motivation therefore formed the basis of the research reported in this thesis.

CHAPTER 3 METHODOLOGY

3.1 Chapter Overview

This chapter discusses the methodology employed to conduct the research. It vividly describes and explains how the developed algorithms work. The Chapter begins with the phases involved in the research, followed by grid scheduling modeling in section 3.2 and ends with simulated algorithms in section 3.3.

3.2 Research overview and Process Flow

Figure 3.1 gives the phases involved in the research.

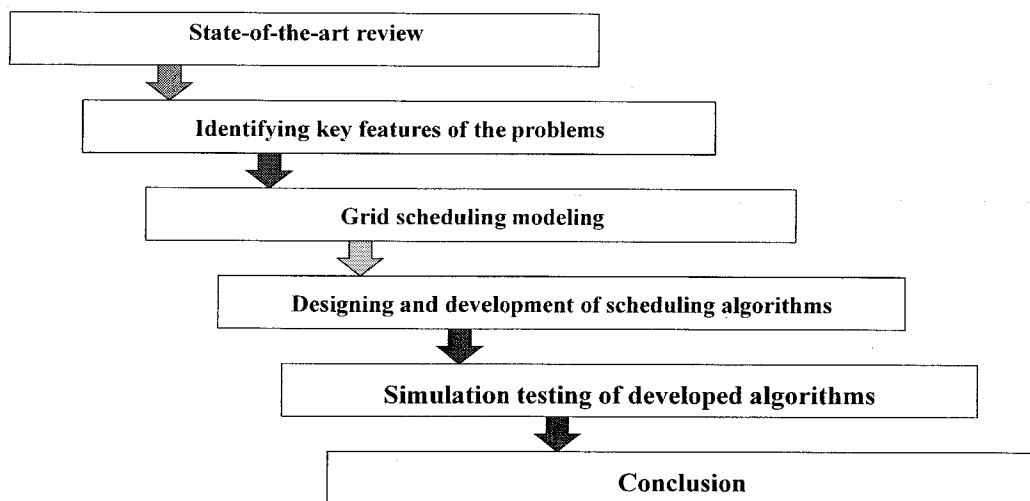


Figure 3.1: Research overview and process flow.

The process flows for the execution of the research are described below.

- A. State-of-the-Art Review: Several factors are considered related to scheduling. For example, scheduling policies, job workflow, job priority, job queue management policies, time and space constraints and deadlines. Therefore, as a first step in achieving our research objectives, we carried out the literature review and we highlighted the current mechanisms and practices for grid scheduling and resource allocation.
- B. Identification of Key Features of the Problem: First, the key characteristics of grid scheduling and deadlines were identified. This was followed by identification of pilot scenarios to assess the impact of jobs deadline on the performance of different grid scheduling strategies.
- C. Grid Scheduling Modeling: The grid system was thereafter modelled as a network of geographically distributed computing sites, where each site itself consists of a number of jobs. In focus were on the dynamics of the model that are relevant to scheduling and robustness. The research assumed that a large number of users are participating in the grid environment and each one is associated with a particular site. We supposed that each user submits a job and that each job would be executed by different nodes.
- D. Designing and Development of Scheduling Algorithms: next was the development and design of some grid scheduling algorithms by using intelligent optimization techniques including constraint programming.
- E. Simulation Testing of Developed Algorithms: The proposed solution was integrated in an experimental grid. The performance of the grid scheduling algorithms on the basis of defined objectives was evaluated.
- F. Conclusion: Here, conclusion of research findings with some future recommendations was drawn in agreement with the objectives of the research.

3.3 Grid Scheduling Modeling

A suitable scheduling model comprises features shown in Figure 3.2.

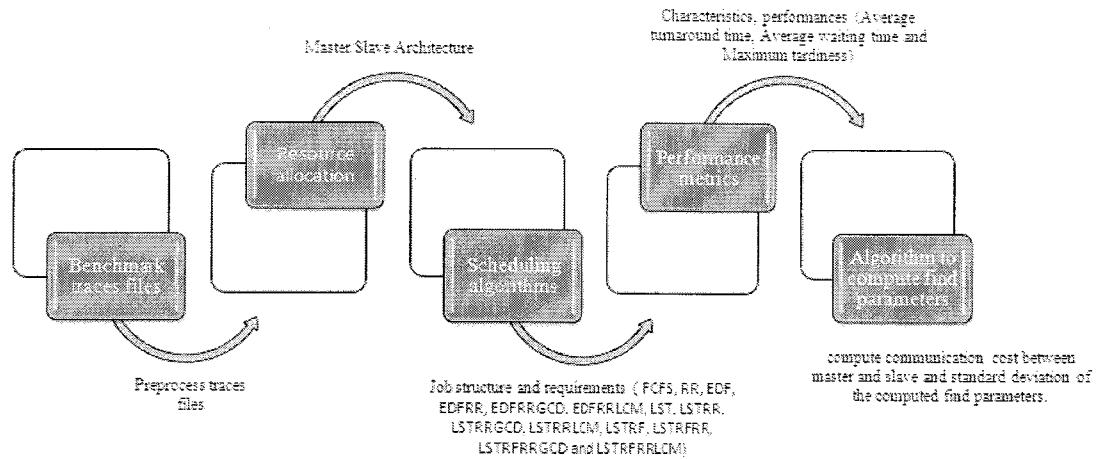


Figure 3.2: Scheduling modeling

3.3.1 Benchmark Traces files

The traces files used were downloaded from grid workloads archive which offer (anonymized) workload traces from grid infrastructure to scientists and also to professionals alike [80]. This analysis was carried out using all trace files provided by grid workload archive which are as follows:

- i. DAS-2 traces: was provided by the advanced school for computing and imaging, the owner of the DAS-2 system [81].
- ii. Grid'5000 traces: was provided by the Grid'5000 team (Dr. Franck Cappello and Dr. Olivier Richard), the owners of the Grid'5000 system, and by the OAR team [82].
- iii. NorduGrid traces: was provided by the nordugrid team (Dr. Balasz Konya), the owners of the Nordugrid system [83].
- iv. AuverGrid traces: was provided by the auvergrid team (Dr. Emmanuel Medernach), the owners of the Auvergrid system [84].
- v. SHARCNET traces: was provided by John Morton and Clayton Chrusch, who also helped with background information and interpretation in high performance computing [85].
- vi. LCG traces: was provided by the e-Science group of HEP, at Imperial

College London, and made publicly available by Hui Li through the parallel workloads archive [86].

Moreover, the entire trace file contains only two parameters- arrival time and burst time. As for deadline parameter, we had to pre-process all traces files and generate deadline parameter using Monte Carlo distribution methods [87]. Monte Carlo methods are a set of computational algorithms that rely on repeated random sampling to compute results. Usually, these methods are mostly used for calculation by a computer and tend to be used when it is infeasible to compute an exact result with a deterministic algorithm.

3.3.2 Resource Allocation

The master-slave architecture was used for testing the developed scheduling algorithms, as shown in Figure 3.3. This involves the use of an actual cluster. The master takes processes as input and distributes the processes on the cluster processors using a simple allocation strategy for parallel computation. In this case, all workload traces are used as input. The total number of jobs is divided by the number of processors, and those numbers of jobs are distributed to each slave where the scheduling algorithms are executed for computation.

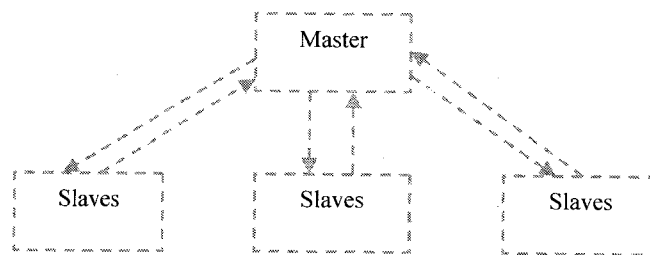


Figure 3.3: Master/Slave Architecture

Each slave receives job, described by its process ID, arrival time, burst time and deadline.

3.3.3 Scheduling Algorithms

This is defining the way in which tasks are assigned to resources. The primary requirement associated with the scheduling policy could possibly be the practical applicability.

In this research, fourteen (14) different versions of scheduling algorithms were simulated in a real computational grid environment. These include:

- Earliest Deadline First Based Round Robin (EDFRR)
- Earliest Deadline First Based Round Robin using greatest common divisor of overall burst time rate as time quantum (EDFRRGCD)
- Earliest Deadline First Based Round Robin using lowest common multiple of overall burst time rate as time quantum (EDFRRLCM)
- Least Slack Time Based Round Robin (LSTRR)
- Least Slack Time Based Round Robin using greatest common divisor of overall burst time rate as time quantum (LSTRRGCD)
- Least Slack Time Based Round Robin using lowest common multiple of overall burst time as time quantum (LSTRRLCM)
- Least Slack Time Rate First Based Round Robin (LSTRFRR)
- Least Slack Time Rate First Based Round Robin using greatest common divisor of overall burst time rate as time quantum (LSTRFRRGCD)
- Least Slack Time Rate First Based Round Robin using lowest common multiple of overall burst time as time quantum (STRFRRLCM)
- First Come First Serve (FCFS), Earliest Deadline First (EDF), Least Slack Time Rate First (LSTRF), Least Slack Time (LST) and Round Robin (RR) scheduling algorithms were used as baseline approaches (including the hybrids of these aforementioned scheduling algorithms).

3.3.4 Performance Metrics

For the purpose of measuring the performances of the scheduling algorithms the following metrics were used:

- i. Average turnaround time: Referred to the average time taken between the submission of job for execution and the return of the completed result.
- ii. Average waiting time: Referred to the average waiting time of job before its final execution.
- iii. Maximum tardiness: Referred to the maximum time delay between turnaround time and deadline time.

3.3.5 Algorithm to Compute Find Parameters

A simple algorithm was used to compute communication cost between master and slave, and to compute the standard deviation of the computed aforementioned parameters.

3.4 Scheduling Algorithms

The description of the proposed scheduling algorithms used in the research (FCFS, RR, EDF, EDFRR, EDFRRGCD, EDFRRLCM, LST, LSTRR, LSTRRGCD, LSTRRLCM, LSTRF, LSTRFRR, LSTRFRRGCD and LSTRFRRLCM) is presented.

Before describing the actual algorithms few terminologies used in the algorithms which require some explanations are:

Assume J_i : i th Job;

n : number of jobs;

n_{si} : number of slaves of job i ;

x_i : number of jobs per slave of job i ;

TQ_i : time quantum of job i ;

T_i : arrival time of job i ;
 d_i : deadline of job i ;
 Ed_i : minimum deadline of job i ;
 α_i : burst time of job i ;
 Ea_i : minimum burst time of job i ;
 C_i : Job completion time of job i ;
 D_i : absolute deadline time of job i ;
 T_{REi} : remaining execution time of job i ;
 T_{RDi} : remaining absolute deadline time of job i ;
 Pr_i : priority rate of job i ;
 $LCM(\alpha_1 \text{ to } \alpha_n)$: lowest common multiple of overall burst time of job i ;
 $GCD(\alpha_1 \text{ to } \alpha_n)$: greatest common divisor of overall burst time of job i ;
 T_{TRMi} : master turnaround time of job i ;
 T_{TRS_i} : slave turnaround time of job i ;
 T_{WTMi} : master waiting time of job i ;
 T_{WTS_i} : slave waiting time of job i ;
 R_{ETi} : starting execution time of job i ;
 S_{ETi} : ending execution time of job i ;
 S_{TEi} : slave total execution time of job i ;
 M_{TEi} : master total execution time of job i ;
 T_{CTi} : total communication time of job i ;
 T_{TDi} : time delay of job i ;
 T_{TRDi} : tardiness of job i ;
 T_{Max_TRD} : maximum tardiness;
 Std_T_{Tri} : standard deviation turnaround time of job i ;

Std_T_{WTi} : standard deviation waiting time of job i ;

Std_T_{TRDi} : standard deviation tardiness time of job i ;

S -list: Sorted list;

- I. Number of jobs per slave x_i : refers to the number of jobs per each slave for execution.

$$x_i = n / n_{si} \dots\dots\dots (1)$$

- II. Time delay T_{TDi} : Referred to the time difference between burst time and deadline time.

$$\text{Time delay, } T_{TDi}: d_i - a_i \dots\dots\dots (2)$$

- III. Minimum deadline time Ed_i : sorting jobs based on minimum deadline first.

$$Ed_i \dots\dots\dots (3)$$

- IV. Minimum burst time Ea_i : sorting jobs based on minimum burst time first.

$$Ea_i \dots\dots\dots (4)$$

- V. Time quantum TQ_i : referred to a fixed time for each job to be executed in cyclic manner meaning 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.

$$TQ_i \dots\dots\dots (5)$$

However, taking time quantum as least common multiple (LCM) is by computing the overall burst time rate of jobs, i.e., by grouping the burst time into some priority rate.

$$\text{Time quantum, take } TQ_i = LCM(a_1 \text{ to } a_n) \dots\dots\dots (6)$$

Also, taking time quantum as greatest common divisor (GCD) is by computing the overall burst time rate of jobs, i.e., by grouping the burst time into some priority rate.

$$\text{Time quantum, take } TQ_i = GCD(a_1 \text{ to } a_n) \dots\dots\dots (7)$$

- VI. Absolute deadline: referred to the time within which the execution of a task should be completed.

$$\text{Absolute deadline } D_i = (d_i + r_i) \dots\dots\dots (8)$$

VII. Remaining execution time: referred to the time remain of a job in the process of execution.

$$\text{Remaining execution time } T_{REi} = (\alpha_i - r_i) \dots \dots \dots (9)$$

VIII. Remaining absolute deadline:referred to the remaining deadline time of job in the process of execution.

$$\text{Remaining absolute deadline } T_{RDi} = (d_i + r_i) - r_i \dots \dots \dots (10)$$

IX. Priority rate: it determines the priority of which job to be executed first in d ready queue.

$$\text{Priority rate } Pr_i = \frac{T_{REi}}{T_{RDi} - r_i} \dots \dots \dots (11)$$

X. Total communication time T_{CTi} : refers to total execution time taken for each master or slaves to finish its execution process:

$$\text{Master: } M_{TEi} = R_{ETi} - S_{ETi} \dots \dots \dots (12)$$

$$\text{Slave: } S_{TEi} = R_{ETi} - S_{ETi} \dots \dots \dots (13)$$

$$\text{Therefore: } T_{CTi} = M_{TEi} - S_{TEi} \dots \dots \dots (14)$$

XI. Turnaround time: Referred to the total time taken between the submission of job for execution and the return of the completed result.

$$\text{Slave turnaround time } T_{TRSi} = C_i - T_i \dots \dots \dots (15)$$

$$\text{Master turnaround time } T_{TRMi} = T_{TRSi} + T_{CTi} \dots \dots \dots (16)$$

Average turnaround time,

$$T_{Avg_TR} = \frac{\sum_{i=1}^n T_{TRMi}}{n} \dots \dots \dots (17)$$

Standard deviation,

$$Std_T_{TR} = \sqrt{\frac{\sum (x_i - Avg_T_{TRSi})^2}{(n-1)}} \dots \dots \dots (18)$$

XII. Waiting time: Referred to the total waiting time of job before its final execution.

$$\text{Slave waiting time } T_{WTSi} = T_{TRSi} - \alpha_i \dots \dots \dots (19)$$

$$\text{Master waiting time } T_{WTMi} = T_{WTSi} + T_{CTi} \dots \dots \dots (20)$$

Average waiting time,

$$T_{Avg_WT} = \frac{\sum_{i=1}^n T_{WTMi}}{n} \dots \dots \dots (21)$$

Standard deviation,

$$Std_{-T_{WT}} \sqrt{\frac{\sum (x_i - Avg_{-T_{WTSi}})^2}{(n-1)}} \dots \dots \dots (22)$$

XIII. Maximum tardiness: Referred to the maximum time delay between turnaround time and deadline time.

$$\text{Slave tardiness, } T_{TRDSi} = d_i - T_{TRSi} \dots \dots \dots (23)$$

$$\text{Master tardiness } T_{TRDMi} = T_{TRDMi} + T_{CTMi} \dots \dots \dots (24)$$

$$\text{Maximum Tardiness } T_{Max_TRD} = Max (T_{TRDM1}, T_{TRDM2}, \dots, T_{TRDMn}) \dots \dots \dots (25)$$

Standard deviation,

$$Std_{-T_{TRD}} \sqrt{\frac{\sum (x_i - Avg_{-T_{TRDMi}})^2}{(n-1)}} \dots \dots \dots (26)$$

The master-slave architecture was used in this research for testing the developed scheduling algorithms, as shown in Fig.1. This involves the use of an actual cluster. The master takes processes as the input and distributes the processes on the cluster processors using a simple allocation strategy for parallel computation. Real workload traces, which comprise DAS-2 [81], Grid5000 [82], NorduGrid [83], AuverGrid [84], SHARCNET [85] and LCG [86], are used as input. The total number of jobs is divided by the number of processors, and those numbers of jobs are distributed to each slave, where the scheduling algorithms are executed for computation.

General framework of all the algorithms is as follows:

Begin

Master:

Begin master

Input: pool of jobs with processID, arrival time, burst time and deadline

Compute number of jobs per processors (1)

Distribute number of jobs to slaves for execution of algorithms (1)

.....

Execute scheduling algorithms on slaves (A/B/C/D/E/F/G/H/I/J/K/L/M/N)

.....

.....

Master:

Receive the value of Turnaround Time, value of Waiting Time, value of Tardiness and value of Slave Total Execution Time from each of the slaves

Compute execution time (12)

Using (8) Compute total communication time between master and slaves

Compute the value of Turnaround Time using (16)

Compute the value of Waiting Time using (20)

Compute the value of Tardiness using (24)

Compute the value of Average Turnaround Time using (17)

Compute the value of Average Waiting Time using (21)

Compute the value of Average Tardiness using (25)

Compute the value of Standard Deviation of Turnaround Time using (18)

Compute the value of Standard Deviation of Waiting Time using (22)

Compute the value of Standard Deviation of Tardiness using (26)

End master

End

However, for each and every algorithm execution master computes total turnaround time, total waiting time and total tardiness and then finally the average turnaround time, average waiting time and maximum tardiness value, to identify the maximum time delay of jobs execution.

A. First-Come-First-Served Scheduling Algorithm (FCFS): In this procedure, each

slave will receive job, described by its process ID, arrival time, burst time and deadline. Dispatching processes are based on their arrival time on the ready queue. Once a process has a processor, it will keep running until it finished executing. Once executed, it will be terminated and then the next process will be dispatched from the ready queue. This process will continue until the pool is empty. The value of turnaround time, waiting time and tardiness for each job are computed and returned to master.

The compact algorithm is presented below:

Algorithm FCFS:

Begin

Slave:

 Begin slave

 For all jobs in the pool

 Arrange the job list in ascending order based on FCFS (*S-list*)

 while (*S-list* is not empty)

 Begin

 Execute the job at CPU level based on demand

 Compute the value of Turnaround Time using (15)

 Compute the value of Waiting Time using (19)

 Compute the value of Tardiness using (23)

 Compute execution time (13)

 Return value of Turnaround Time, value of Waiting Time,
 value of Tardiness and value of Slave Total Execution Time
 to master

 Endwhile

 End slave

- B. Round Robin (RR) Scheduling Algorithm: Each slave will receive job, described by its process ID, arrival time, burst time and deadline. The ready queue is preserved as a first come first served (FCFS) queue. Dispatching processes is from the head of the ready queue for execution by the processor. The pre-emption of a process for execution is based on system defined variable, named as time quantum. However, as soon as a process execution is

completed, before its time quantum expired, it will be terminated as well as deleted from the system. Therefore, next process will be dispatched from the ready queue. This process will continue until the pool is empty. The value of turnaround time, waiting time and tardiness for each job are computed and return to master.

The compact algorithm is presented below:

Algorithm RR:

Begin

Slave:

 Begin slave

 For all jobs in the pool

 Time quantum TQ (5)

 Arrange the job list in ascending order based on FCFS (*S-list*)

 while (*S-list* is not empty)

 Begin

 Execute the job at CPU level based on demand

 Compute the value of Turnaround Time using (15)

 Compute the value of Waiting Time using (19)

 Compute the value of Tardiness using (23)

 Compute execution time (13)

 Return value of Turnaround Time, value of Waiting Time,
value of Tardiness and value of Slave Total Execution Time
to master

 if ($a_i > 0$)

 Begin

$a_i - TQ$

 Endif

 Endwhile

 End slave

- C. Earliest Deadline First Scheduling Algorithm (EDF): Here, each slave will receive job, described by its process ID, arrival time, burst time and deadline. Processes are dispatched based on minimum deadline on the ready queue.

When a process has been completed, its task it will be terminated and then the next job with a minimum deadline will be dispatched from the ready queue in a non pre-emptive way. If two tasks have the same absolute deadlines, EDF chooses based on FCFS in order to break the tie. 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. This process will continue until the pool is empty. The value of turnaround time, waiting time and tardiness for each job are computed and return to master.

The compact algorithm is presented below:

Algorithm EDF:

Begin

Slave:

 Begin slave

 For all jobs in the pool

 Arrange the job list in ascending order based on the minimum deadline time as mentioned in criteria III (*S-list*)

 if ($Ed_i = Ed_j$)

 Arrange J_i, J_j based on FCFS

 while (*S-list* is not empty)

 Begin

 Execute the job at CPU level based on demand

 Compute the value of Turnaround Time using (15)

 Compute the value of Waiting Time using (19)

 Compute the value of Tardiness using (23)

 Compute execution time (13)

 Return value of Turnaround Time, value of Waiting Time, value of Tardiness and value of Slave Total Execution Time to master

 Endwhile

 End slave

- D. Earliest Deadline First based Round Robin Scheduling Algorithm (EDFRR): In this procedure, each slave will receive job, described by its process ID, arrival

time, burst time and deadline. Processes are dispatched based on minimum deadline on the ready queue. The pre-emption of a process for execution is based on system defined variable, named as time quantum. However, as soon as a process execution is completed, before its time quantum expired, it will be terminated as well as deleted from the system and then the next job with a minimum deadline will be dispatched from the ready queue. However, if two tasks have the same absolute deadlines, EDF chooses based on FCFS in order to break the tie. This process will continue until the pool is empty. The value of turnaround time, waiting time and tardiness for each job are computed and return to master.

The compact algorithm is presented below:

Algorithm EDFRR:

Begin

Slave:

 Begin slave

 Time quantum TQ (5)

 For all jobs in the pool

 Arrange the job list in ascending order based on the minimum deadline time as mentioned in criteria III (*S-list*)

 if ($Ed_i = Ed_j$)

 Arrange J_i, J_j based on FCFS

 while (*S-list* is not empty)

 Begin

 Execute the job at CPU level based on demand

 Compute the value of Turnaround Time using (15)

 Compute the value of Waiting Time using (19)

 Compute the value of Tardiness using (23)

 Compute execution time (13)

 Return value of Turnaround Time, value of Waiting Time, value of Tardiness and value of Slave Total Execution Time to master

 if ($\alpha_i > 0$)


```

Begin
     $a_i - TQ$ 
Endif
Endwhile
End slave

```

- E. Earliest Deadline First based Round Robin Scheduling Algorithm (EDFRRGCD): Each slave will receive job, described by its process ID, arrival time, burst time and deadline. However, it assigns time quantum, by computing the GCD of all burst time rate, then sorting job based on minimum deadline on the ready queue. Processes are dispatched based on minimum deadline on the ready queue. The pre-emption of a process for execution is based on system defined variable, named as time quantum. However, as soon as a process execution is completed, before its time quantum expired, it will be terminated as well as deleted from the system and then the next job with a minimum deadline will be dispatched from the ready queue. However, if two tasks have the same absolute deadlines, EDF chooses based on FCFS in order to break the tie. This process will continue until the pool is empty. The value of turnaround time, waiting time and tardiness for each job are computed and return to master.

The compact algorithm is presented below:

Algorithm EDFRRGCD:

Begin

Slave:

Begin slave

Compute Time quantum TQ (7)

For all jobs in the pool

Arrange the job list in ascending order based on the minimum deadline time as mentioned in criteria III (*S-list*)

if ($Ed_i = Ed_j$)

Arrange J_i, J_j based on FCFS

while (*S-list* is not empty)

Begin

```

Execute the job at CPU level based on demand
Compute the value of Turnaround Time using (15)
Compute the value of Waiting Time using (19)
Compute the value of Tardiness using (23)
Compute execution time (13)
Return value of Turnaround Time, value of Waiting Time,
value of Tardiness and value of Slave Total Execution Time
to master
if ( $\alpha_i > 0$ )
  Begin
     $\alpha_i - TQ$ 
    Compute Time quantum  $TQ$  (7)
  Endif
Endwhile
End slave

```

- F. Earliest Deadline First based Round Robin Scheduling Algorithm (EDFRRLCM): Here, each slave will receive job, described by its process ID, arrival time, burst time and deadline. However, it assigns time quantum, by computing the LCM of all burst time rate, then sorting job based on minimum deadline on the ready queue. Processes are dispatched based on minimum deadline on the ready queue. The pre-emption of a process for execution is based on system defined variable, named as time quantum. However, as soon as a process execution is completed, before its time quantum expired, it will be terminated as well as deleted from the system and then the next job with a minimum deadline will be dispatched from the ready queue. However, if two tasks have the same absolute deadlines, EDF chooses based on FCFS in order to break the tie. This process will continue until the pool is empty. The value of turnaround time, waiting time and tardiness for each job are computed and return to master.

The compact algorithm is presented below:

Algorithm PDSA:

Begin

Slave:

Begin slave

Compute Time quantum TQ (6)

For all jobs in the pool

Arrange the job list in ascending order based on the minimum deadline time as mentioned in criteria III (*S-list*)

if ($Ed_i = Ed_j$)

Arrange J_i, J_j based on FCFS

while (*S-list* is not empty)

Begin

Execute the job at CPU level based on demand

Compute the value of Turnaround Time using (15)

Compute the value of Waiting Time using (19)

Compute the value of Tardiness using (23)

Compute execution time (13)

Return value of Turnaround Time, value of Waiting Time, value of Tardiness and value of Slave Total Execution Time to master

if ($\alpha_i > 0$)

Begin

$\alpha_i - TQ$

Compute Time quantum TQ (6)

Endif

Endwhile

End slave

- G. Least Slack Time Scheduling Algorithm (LST): Here, each slave will receive job, described by its process ID, arrival time, burst time and deadline, then compute the value of time delay for each job by sorting out the jobs on the basis of minimum time delay in ascending order. Moreover, the algorithm selects the jobs with minimum time delay for execution. If multiple jobs have same time delay value then, it will break the tie by selecting a job from job set on the basis of FCFS and then execute the job at CPU level for its given burst time (i.e.

Demand) in a non pre-emptive way. This process will continue until the pool is empty. The value of turnaround time, waiting time and tardiness for each job are computed and return to master.

The compact algorithm is presented below:

Algorithm LST:

Begin

Slave:

 Begin slave

 For all jobs in the pool

 Compute the time delay of all processes using (2)

 Arrange the job list in ascending order based on the minimum time delay as mentioned in criteria II (*S-list*)

 if ($T_{TDi} = T_{TDj}$)

 Arrange J_i, J_j based on FCFS

 while (*S-list* is not empty)

 Begin

 Execute the job at CPU level based on demand

 Compute the value of Turnaround Time using (15)

 Compute the value of Waiting Time using (19)

 Compute the value of Tardiness using (23)

 Compute execution time (13)

 Return value of Turnaround Time, value of Waiting Time,

 value of Tardiness and value of Slave Total Execution Time

 to master

 Endwhile

 End slave

- H. Least Slack Time based Round Robin Scheduling Algorithm (LSTRR): processes are executed in this algorithm with the closest deadline time delay in the cyclic manner using a dynamic time quantum. The slaves take the input from master, whereas each job is described by its process ID, arrival time, burst time and deadline. In executing the algorithm, time quantum is given as a fixed value, and then it computes the value of time delay for each job by sorting out

the jobs on the basis of minimum time delay in ascending order, then selecting the jobs with minimum time delay in execution. If multiple jobs have same time delay value then, it will break the tie by selecting a job from job set on the basis of FCFS. Processes are dispatched based on minimum time delay on the ready queue. The pre-emption of a process for execution is based on system defined variable, named as time quantum. However, as soon as a process execution is completed, before its time quantum expired, it will be terminated as well as deleted from the system and then the next process is then dispatched from the head of the ready queue. This process will continue until the pool is empty. The value of turnaround time, waiting time and tardiness for each job are computed and return to master.

The compact algorithm is presented below:

Algorithm LSTRR:

Begin

Slave:

Begin slave

Time quantum TQ (2)

For all jobs in the pool

Compute the time delay of all processes using (2)

Arrange the job list in ascending order based on the minimum time delay as mentioned in criteria II (*S-list*)

if ($T_{TDi} = T_{TDj}$)

Arrange J_i, J_j based on FCFS

while (*S-list* is not empty)

Begin

Execute the job at CPU level based on demand

Compute the value of Turnaround Time using (15)

Compute the value of Waiting Time using (19)

Compute the value of Tardiness using (23)

Compute execution time (13)

```

Return value of Turnaround Time, value of Waiting Time,
value of Tardiness and value of Slave Total Execution Time
to master
if ( $\alpha_i > 0$ )
  Begin
     $\alpha_i - TQ$ 
  Endif
Endwhile
End slave

```

I. Least Slack Time based Round Robin Scheduling Algorithm (LSTRRGCD):

This algorithm executes the process with the closest deadline time delay in the cyclic manner using a dynamic time quantum. During execution, the slaves take the input from master, whereas each job is described by its process ID, arrival time, burst time and deadline. It assigns time quantum, by computing the GCD of all burst time rate, and then compute the value of time delay for each job by sorting out the jobs on the basis of time delay in ascending order, then selecting the jobs with minimum time delay in execution. If multiple jobs have same time delay value then, it will break the tie by selecting a job from job set on the basis of FCFS. Processes are dispatched based on minimum time delay on the ready queue. The pre-emption of a process for execution is based on system defined variable, named as time quantum. However, as soon as a process execution is completed, before its time quantum expired, it will be terminated as well as deleted from the system and then the next process is then dispatched from the head of the ready queue. This process will continue until the pool is empty. The value of turnaround time, waiting time and tardiness for each job are computed and return to master.

The compact algorithm is presented below:

Algorithm LSTRRGCD:

Begin

Slave:

Begin slave

Compute time quantum TQ (7)

For all jobs in the pool
 Compute the time delay of all processes using (2)
 Arrange the job list in ascending order based on the minimum time delay
 as mentioned in criteria II (*S-list*)
 if ($T_{TDi} = T_{TDj}$)
 Arrange J_i, J_j based on FCFS
 while (*S-list* is not empty)
 Begin
 Execute the job at CPU level based on demand
 Compute the value of Turnaround Time using (15)
 Compute the value of Waiting Time using (19)
 Compute the value of Tardiness using (23)
 Compute execution time (13)
 Return value of Turnaround Time, value of Waiting Time,
 value of Tardiness and value of Slave Total Execution Time
 to master
 if ($\alpha_i > 0$)
 Begin
 $\alpha_i - TQ$
 Compute time quantum TQ (7)
 Endif
 Endwhile
 End slave

- J. Least Slack Time based Round Robin Scheduling Algorithm (LSTRRLCM): In this process, the slaves take the input from master, where as each job is described by its process ID, arrival time, burst time and deadline. It assigns time quantum, by computing LCM of all burst rate time, and then compute the value of time delay for each job by sorting out the jobs on the basis of time delay in ascending order, then selecting the jobs with minimum time delay in execution. If multiple jobs have same time delay value then, it will break the tie by selecting a job from job set on the basis of FCFS. Processes are dispatched based on minimum time delay on the ready queue. The pre-emption

of a process for execution is based on system defined variable, named as time quantum. However, as soon as a process execution is completed, before its time quantum expired, it will be terminated as well as deleted from the system and then the next process is then dispatched from the head of the ready queue. The value of turnaround time, waiting time and tardiness for each job are computed and return to master.

The compact algorithm is presented below:

Algorithm LSTRRLCM:

Begin

Slave:

 Begin slave

 Compute time quantum TQ (6)

 For all jobs in the pool

 Compute the time delay of all processes using (2)

 Arrange the job list in ascending order based on criteria II (*S-list*)

 if ($T_{TDi} = T_{TDj}$)

 Arrange J_i, J_j based on FCFS

 while (*S-list* is not empty)

 Begin

 Execute the job at CPU level based on demand

 Compute the value of Turnaround Time using (15)

 Compute the value of Waiting Time using (19)

 Compute the value of Tardiness using (23)

 Compute execution time (13)

 Return value of Turnaround Time, value of Waiting Time,
 value of Tardiness and value of Slave Total Execution Time
 to master

 if ($\alpha_i > 0$)

 Begin

$\alpha_i - TQ$

 Compute time quantum TQ (6)

 Endif

Endwhile

End slave

- K. Least Slack Time Rate First Scheduling Algorithm (LSTRF): This algorithm determines the priority rate of job execution by computing the value of the slack time priority rate value of which job to be executed first in the ready queue. However, the slaves take the input from master, whereas each job is described by its process ID, arrival time, burst time and deadline, then compute the value absolute deadline, then the value of remaining execution time, the value remaining absolute deadline and then compute the value of the priority rate for each job by sorting out the jobs on the basis of priority rate in ascending order, then selecting the jobs with the minimum priority rate for execution. If multiple jobs have same priority rate value then, it will break the tie by selecting a job from job set on the basis of FCFS. It then executes the job at CPU level for its given burst time (i.e., Demand) in a non pre-emptive way. This process will continue until the pool is empty. The value of turnaround time, waiting time and tardiness for each job are computed and return to master.

The compact algorithm is presented below:

Algorithm LSTRF:

Begin

Slave:

Begin slave

For all jobs in the pool

Compute the value absolute deadline (8)

Compute the value of remaining execution time (9)

Compute the value remaining absolute deadline (10)

Compute the value of the priority rate (11)

Arrange the job list in ascending order based on minimum priority rate as mentioned in criteria IX (S-list)

if ($Pr_i = Pr_j$)

Arrange J_i, J_j based on FCFS

while (*S-list* is not empty)

Begin

Execute the job at CPU level based on demand
 Compute the value of Turnaround Time using (15)
 Compute the value of Waiting Time using (19)
 Compute the value of Tardiness using (23)
 Compute execution time (13)
 Return value of Turnaround Time, value of Waiting Time,
 value of Tardiness and value of Slave Total Execution Time
 to master

Endwhile

End slave

- L. Least Slack Time Rate First with Round Robin Scheduling Algorithm (LSTRRR): The slaves take the input from master, whereas each job is described by its process ID, arrival time, burst time and deadline. However, time quantum is given as a fixed value, and then computes the value absolute deadline, then the value of remaining execution time, the value remaining absolute deadline and then compute the value of the priority rate for each job by sorting out the jobs on the basis of priority rate in ascending order, then selecting the jobs with the minimum priority rate for execution. If multiple jobs have same priority rate value then, it will break the tie by selecting a job from job set on the basis of FCFS. Processes are dispatched based on minimum priority rate on the ready queue. The pre-emption of a process for execution is based on system defined variable, named as time quantum. However, as soon as a process execution is completed, before its time quantum expired, it will be terminated as well as deleted from the system and then the next process is then dispatched from the head of the ready queue. This process will continue until the pool is empty. The value of turnaround time, waiting time and tardiness for each job are computed and return to master.

The compact algorithm is presented below:

Algorithm LSTRFRR:

Begin

Slave:

Begin slave

Time quantum TQ (2)

For all jobs in the pool

 Compute the value absolute deadline (8)

 Compute the value of remaining execution time (9)

 Compute the value remaining absolute deadline (10)

 Compute the value of the priority rate (11)

 Arrange the job list in ascending order based on minimum priority rate as mentioned in criteria IX (S-list)

 if ($Pr_i = Pr_j$)

 Arrange J_i, J_j based on FCFS

 while (*S-list* is not empty)

 Begin

 Execute the job at CPU level based on demand

 Compute the value of Turnaround Time using (15)

 Compute the value of Waiting Time using (19)

 Compute the value of Tardiness using (23)

 Compute execution time (13)

 Return value of Turnaround Time, value of Waiting Time, value of Tardiness and value of Slave Total Execution Time to master

 if ($\alpha_i > 0$)

 Begin

$\alpha_i - TQ$

 Endif

 Endwhile

 Endslave

- M. Least Slack Time Rate First based Round Robin Scheduling Algorithm (LSTRFRRGCD): This is similar to the previous algorithm; the slaves take the input from master, where as each job is described by its process ID, arrival time, burst time and deadline. It assigns time quantum, by computing the GCD of all burst time, then compute the value absolute deadline, then the value of remaining execution time, the value remaining absolute deadline and then compute the

value of the priority rate for each job by sorting out the jobs on the basis of priority rate in ascending order, then selecting the jobs with the minimum priority rate for execution. If multiple jobs have same priority rate value then, it will break the tie by selecting a job from job set on the basis of FCFS. Processes are dispatched based on minimum priority rate on the ready queue. The pre-emption of a process for execution is based on system defined variable, named as time quantum. However, as soon as a process execution is completed, before its time quantum expired, it will be terminated as well as deleted from the system and then the next process is then dispatched from the head of the ready queue. This process will continue until the pool is empty. The value of turnaround time, waiting time and tardiness for each job are computed and return to master.

The compact algorithm is presented below:

Algorithm LSTRFRRGCD:

Begin

Slave:

 Begin slave

 Compute time quantum TQ (7)

 For all jobs in the pool

 Compute the value absolute deadline (8)

 Compute the value of remaining execution time (9)

 Compute the value remaining absolute deadline (10)

 Compute the value of the priority rate (11)

 Arrange the job list in ascending order based on minimum priority rate as mentioned in criteria IX (S-list)

 if ($Pr_i = Pr_j$)

 Arrange J_i, J_j based on FCFS

 while (*S-list* is not empty)

 Begin

 Execute the job at CPU level based on demand

 Compute the value of Turnaround Time using (15)

 Compute the value of Waiting Time using (19)

```

    Compute the value of Tardiness using (23)
    Compute execution time (13)
    Return value of Turnaround Time, value of Waiting Time,
    value of Tardiness and value of Slave Total Execution Time
    to master
    if ( $\alpha_i > 0$ )
        Begin
             $\alpha_i - TQ$ 
            Compute time quantum  $TQ$  (7)
        Endif
    Endwhile
End slave

```

N. Least Slack Time Rate First with Round Robin Scheduling Algorithm (LSTRFRRLCM): Similarly, the slaves take the input from master, where as each job is described by its processID, arrival time, burst time and deadline. It assigns time quantum, by computing the LCM of all burst time, then compute the value absolute deadline, then the value of remaining execution time, the value remaining absolute deadline and then compute the value of the priority rate for each job by sorting out the jobs on the basis of priority rate in ascending order, then selecting the jobs with the minimum priority rate for execution. If multiple jobs have same priority rate value then, it will break the tie by selecting a job from job set on the basis of FCFS. Processes are dispatched based on minimum priority rate on the ready queue. The preemption of a process for execution is based on system defined variable, named as time quantum. However, as soon as a process execution is completed, before its time quantum expired, it will be terminated as well as deleted from the system and then the next process is then dispatched from the head of the ready queue. This process will continue until the pool is empty. The value of turnaround time, waiting time and tardiness for each job are computed and return to master.

The compact algorithm is presented below:

Algorithm LSTRFRRLCM:

Begin

Slave:

Begin slave

Compute time quantum TQ (6)

For all jobs in the pool

Compute the value absolute deadline (8)

Compute the value of remaining execution time (9)

Compute the value remaining absolute deadline (10)

Compute the value of the priority rate (11)

Arrange the job list in ascending order based on minimum
priority rate as mentioned in criteria IX (S-list)

if ($Pr_i = Pr_j$)

Arrange J_i, J_j based on FCFS

while (S -list is not empty)

Begin

Execute the job at CPU level based on demand

Compute the value of Turnaround Time using (15)

Compute the value of Waiting Time using (19)

Compute the value of Tardiness using (23)

Compute execution time (13)

Return value of Turnaround Time, value of Waiting Time,
value of Tardiness and value of Slave Total Execution Time
to master

if ($\alpha_i > 0$)

Begin

$a_i - TQ$

Compute time quantum TQ (6)

Endif

Endwhile

End slave

3.5 Experimental Procedures

The procedures adopted in evaluating the algorithms (including their hybrids) which were used in this research are presented as follows:

3.5.1 Experimental Setup

Proper resources have been selected in the first step and experiments were carried out using the facilities of high performance computing centre (HPCC) at Universiti Teknologi PETRONAS. This involves the use of an actual cluster; the master takes process as the input and distributes the processes on the cluster processors using a simple allocation strategy for parallel computation. We incorporated scalability test of scheduling algorithms under an increasing real workload. All experiments were performed by varying the number of processors from 32, 64 and 128, showing the demands of the user's jobs, each with different characteristics.

All algorithms were developed using Java programming language and MPJ, which is a free Java message passing library that enables application designers to create and execute parallel programs for multi-core processors.

3.5.2 Parameter Settings

Prior to the primary experimentation to determine the performance of the scheduling algorithms with respect to benchmarks, preliminary experiments were conducted to traces files, in order to setup the values of deadline parameters. These were initially lacking in the entire traces files. All traces files were and generated deadline parameter pre-processed using Monte Carlo distribution methods [87]. Monte Carlo methods are a set of computational algorithms that rely on repeated random sampling to compute results. These methods are mostly used for calculations by a computer and tend to be used when it is infeasible to compute an exact result with a deterministic algorithm. Also the jobs with negative burst time are excluded from the trace files.

3.5.3 Benchmark Description

A total of six (6) benchmarks traces files were used in subsequent section to perform comparative analysis of the algorithms. These traces file were obtained from grid workloads archive which provided anonymized workload traces from grid environments to researchers and to practitioners alike [80]. However, we carried out our analysis by using all traces files provided by grid workload archive which are as follows:

- i. DAS-2 traces: a wide-area distributed system consisting of 400 CPUs located at five Dutch Universities. DAS-2 is a research test bed, with the workload composed of a large variety of applications. DAS-2 was provided by the advanced school for computing and imaging, the owner of the DAS-2 system [81].
- ii. Grid'5000 traces: Grid'5000 is an experimental grid platform composed of 9 sites geographically distributed in France. Each site comprises one or more groupings, inside Grid'5000. It was made available from the Grid'5000 team (Dr. Franck Cappello and Dr. Olivier Richard), the proprietors from the Grid'5000 system, by the OAR team. The traces collected from Grid5000 include programs in the regions of physics, biomed, math, chemistry, climate, astronomy, language, existence, finance, etc. Additionally, the Grid5000 traces include experimental programs for parallel and distributed systems research [82].
- iii. NorduGrid traces: NorduGrid is really a production grid for academic scientists in Denmark, Estonia, Finland, Norway, Sweden, etc. Since 2002, NorduGrid has been around continuous operation and development, and also, since 2003 industrial, scientific or private organizations with curiosity about grid computing happen to be asked to lead their compute energy towards the NorduGrid as collaborators. It was made available by the NorduGrid team (Dr. Balasz konya), the owners of the NorduGrid system. The traces collected from NorduGrid include programs in the regions of CAS, chemistry, graphics, biomed, and HEP [83].
- iv. AuverGrid traces: AuverGrid is a production grid platform composed of 5

groupings situated geographically within the Auvergne region, France. Groupings are comprised of dual 3GHz Pentium-IV Xeons nodes running Scientific Linux, LPC means "Laboratoire p Physique Corpusculaire" (Laboratory of Corpuscular Physics) of Université Blaise-Pascal, Clermont-Ferrand, France. The Auvergrid project is really a regional grid area of the EGEE project (Enabling Grids for E-science in Europe). It's used mostly for biomedical and-energy physics research. Auvergrid was provided by the Auvergrid team (Dr. Emmanuel Medernach), the owners of the Auvergrid system [84].

- v. SHARCNET traces: SHARCNET is structured like a "cluster of clusters" across south western, central and northern Ontario, made to satisfy the computational needs of scientists inside a diverse quantity of research areas and also to facilitate the introduction of leading-edge tools for top performance computing. This trace consists of up to and including year's price of accounting records in the SHARCNET groupings installed at a number of schools in Ontario, Canada. SHARCNET was provided by John Morton and Clayton Chrusch, who also helped with background information and interpretation in high performance computing [85].
- vi. LCG traces: This log consists of 11 times of activity from multiple nodes that comprise the LCG (Large Hadron Collider Computing Grid). Customers submit serial or parallel jobs to resource brokers. The resource brokers find appropriate assets for undertaking the computation, and send systems for execution around the different systems. LCG was provided by the e-Science group of HEP, at Imperial College London, and made publicly available by hui li through the parallel workloads archive [86].

All algorithms were developed using Java programming language and MPJ, which is a free Java message passing library that enables application designers to create and execute parallel programs for multi-core processors.

3.6 Summary

This chapter described the overall framework of the research methodology used. It incorporated the actual plans which were adopted in order to satisfy the proposed research objectives. In addition, this chapter served as the guidance towards the comparison of various scheduling algorithms. The research comparison analyzes the aforementioned simulated algorithms, the findings of which would be discussed in greater the next chapter.

CHAPTER 4

RESULT AND DISCUSSION

4.1 Chapter Overview

This chapter discusses the results which were obtained from the experimentation in a real computational grid environment. The results analysis was based on the following performance metrics:

This chapter discusses the results which were obtained from the experimentation in a real computational grid environment. The results analysis was based on the following performance metrics:

- Average Turnaround Time
- Average Waiting Time
- Maximum Tardiness

Also the standard deviations of the above performance metrics are calculated.

4.2 Comparative Performance Analysis of Scheduling Algorithms

With parameter settings of benchmark traces files determined, the next phase involved testing the algorithms on benchmark traces files. A total of six (6) traces files with almost different processing demands (two (2) of the traces files are comparatively heavier than the rest four (4)) were used to observe the performance of the developed scheduling algorithms by varying the number of jobs and processors. Therefore, there was need to incorporate scalability test of scheduling algorithms under an increasing real workload, which led to the creation of ten (10) data sets

formed by using 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 98% of the traces files workload, 100000, 200000, 300000, 400000, 500000, 600000, 700000, 800000, 900000 and 1000000 processes, respectively. Also, the number of processors were varied by increasing it with 100%, respectively i.e., from 32 to 64 and 128 processors.

4.2.1 Das-2 Traces

The experiment was carried out using the entire workload of the traces file.

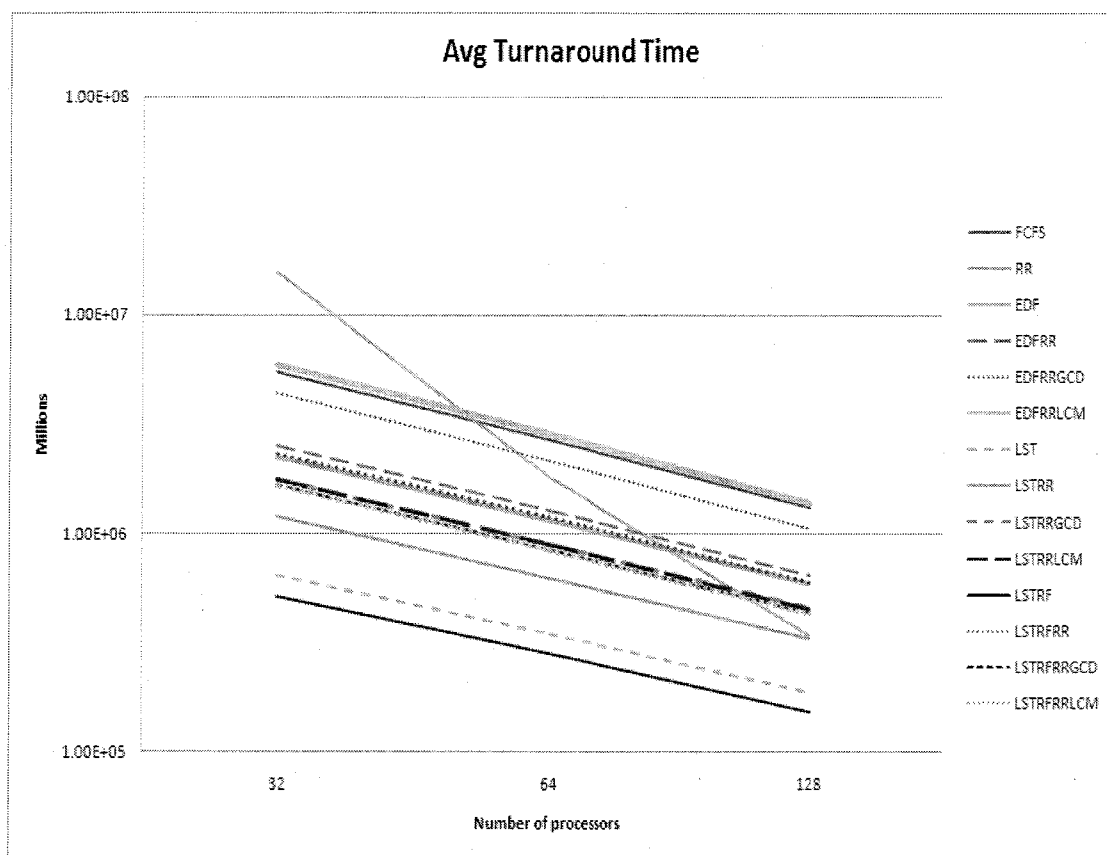


Figure 4.1: Average Turnaround Time

This experiment was carried out by varying the number of processors from 32, 64 and 128 numbers of processors. Based on the observation of Figure 4.1, it is clear that LSTRF, LST, RR, LSTRR, LSTRRGCD, LSTRRLCM, EDFRR, EDFRRGCD, EDFRRLCM, LSTRFRR, LSTRFRRGCD, LSTRFRRLCM and FCFS are smooth and steady from 32 to 64 and 128 processors. EDF showed a sharp fall when number

of processors varied from 32 to 64 and 128 processors. Results showed that LSTRF and LST have the best performance while EDFRRLCM and FCFS showed the worst performance.

To ensure that the value obtained is consistent, standard deviation of each algorithm's turnaround time for each set of experiments, based on 32, 64 and 128 processors were computed. Table 4.1 showed the standard deviation of the experiments whose results are depicted in Figure 4.1.

Table 4.1: Standard Deviation

Standard Deviation	FCFS	RR	EDF	EDFRR	EDFRRGCD	EDFRRLCM	LST	LSTRR	LSTRRGCD	LSTRRLCM	LSTRF	LSTRFRR	LSTRFRRGCD	LSTRFRRLCM
32	2.38E+08	5.23E+07	6.92E+08	1.10E+08	1.93E+08	2.59E+08	2.81E+07	9.89E+07	7.73E+07	7.72E+07	2.26E+07	1.02E+08	7.35E+07	7.24E+07
64	1.71E+08	3.94E+07	1.14E+08	8.01E+07	1.38E+08	1.80E+08	2.19E+07	7.28E+07	5.87E+07	5.63E+07	1.78E+07	7.45E+07	5.37E+07	5.30E+07
128	1.18E+08	3.00E+07	3.05E+07	5.84E+07	9.51E+07	1.24E+08	1.69E+07	5.30E+07	4.11E+07	4.08E+07	1.38E+07	5.44E+07	3.90E+07	3.85E+07

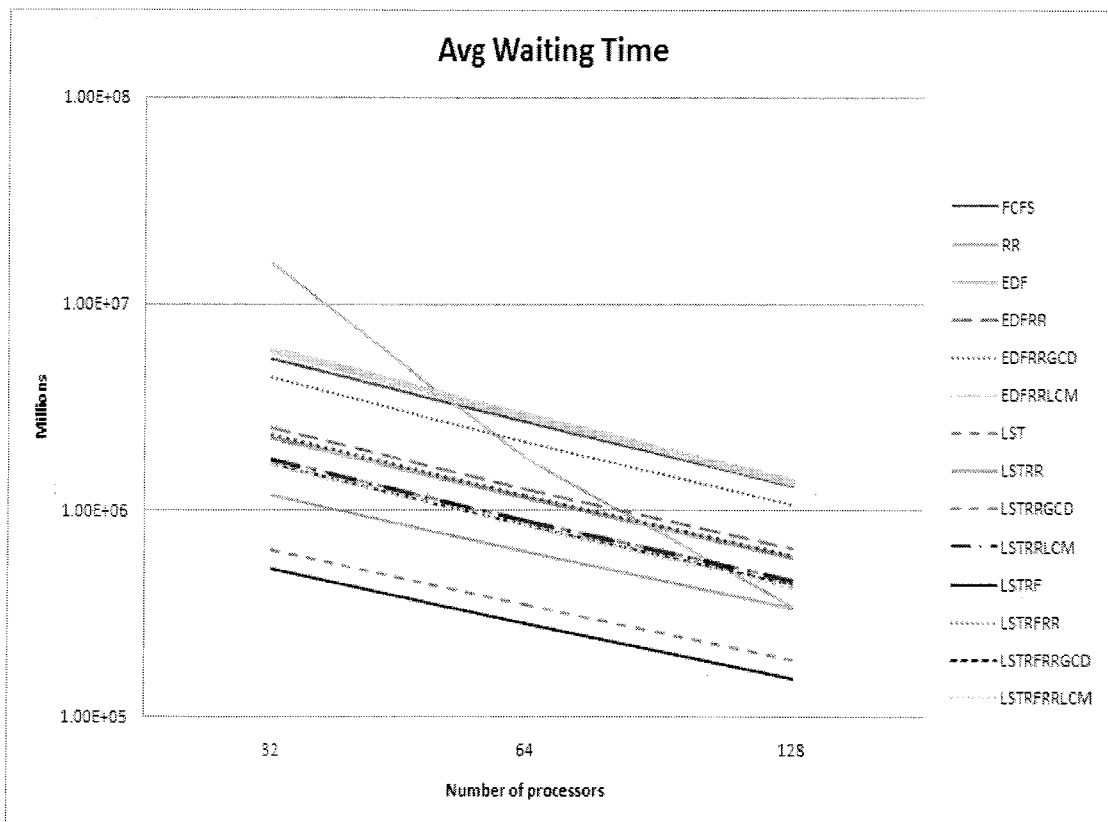


Figure 4.2: Average Waiting Time

It can be observed from Figure 4.2 that LSTRF, LST, RR, LSTRR, LSTRRGCD,

LSTRRLCM, EDFRR, EDFRRGCD, EDFRRLCM, LSTRFRR, LSTRFRRGCD, LSTRFRRLCM and FCFS are smooth and steady from 32 to 64 and 128 processors. EDF showed a sharp fall when number of processors varied from 32 to 64 and 128 processors. Results showed that LSTRF and LST have the best EDFRRLCM and FCFS showed the worst performance.

To ensure that the value obtained is consistent, standard deviation of each algorithm and turnaround time of each set of experiments were computed based on 32, 64 and 128 processors. Table 4.2, showed the standard deviation of Figure 4.2 results.

Table 4.2: Standard Deviation

Standard Deviation	FCFS	RR	EDF	EDFRR	EDFRRGCD	EDFRRLCM	LST	LSTRR	LSTRRGCD	LSTRRLCM	LSTRF	LSTRFRR	LSTRFRRGCD	LSTRFRRLCM
32	2.39E+08	5.23E+07	6.92E+06	1.10E+08	1.18E+06	2.58E+08	2.81E+07	9.98E+07	7.73E+07	7.72E+07	2.26E+07	1.02E+08	7.34E+07	7.24E+07
64	1.71E+08	3.94E+07	1.14E+06	8.01E+07	1.38E+06	1.80E+08	2.19E+07	7.26E+07	5.67E+07	4.07E+07	1.77E+07	7.45E+07	5.38E+07	5.29E+07
128	1.18E+08	3.00E+07	3.06E+07	5.84E+07	9.50E+07	1.24E+08	1.68E+07	5.29E+07	4.11E+07	2.92E+10	1.37E+07	5.43E+07	3.90E+07	3.84E+07

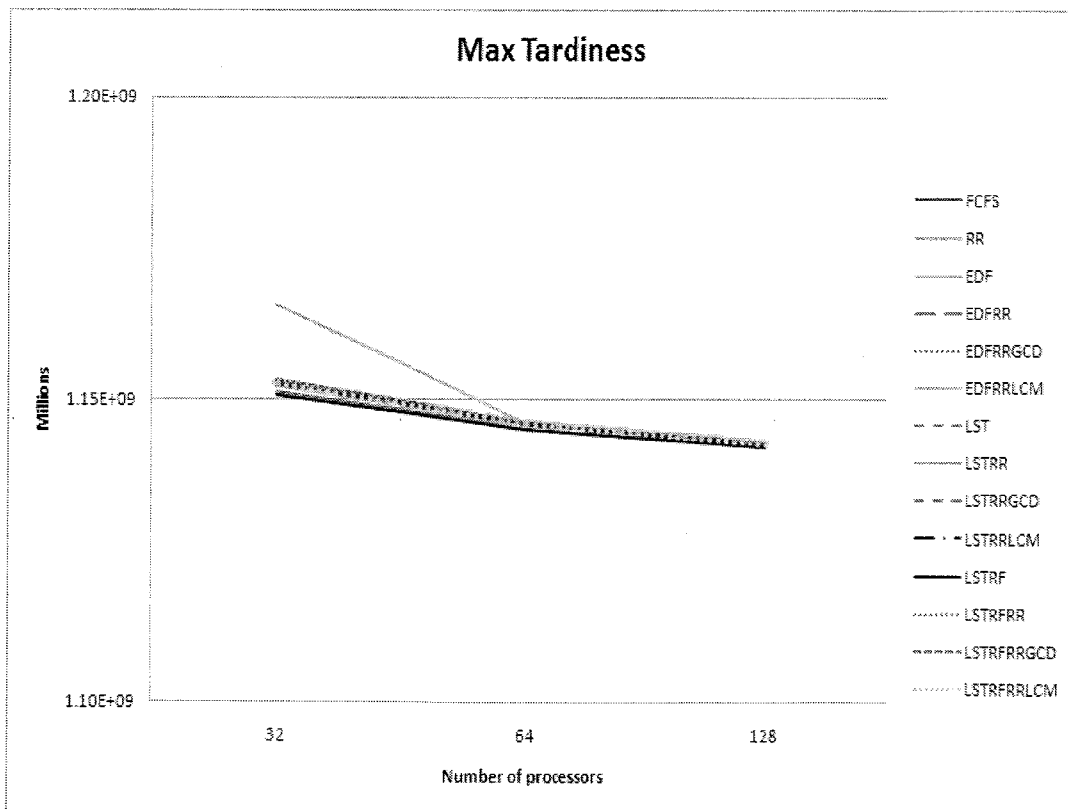


Figure 4.3: Maximum Tardiness

Figure 4.3 show that maximum tardiness is not fixed; rather, it varies with the workload. However, it can be observed that LSTRF, LST, RR, LSTRR, LSTRRGCD, LSTRRLCM, EDFRR, EDFRRGCD, EDFRRLCM, LSTRFRR, LSTRFRRGCD, LSTRFRRLCM and FCFS are smooth and steady as well as overlap one another from 32 to 64 and 128 processors. However, there is a slight difference between the algorithms. EDF showed a slight fall from 32 to 64 processors and from 64 to 128 processors. Results showed that LSTRF and LST have the best EDFRRLCM and EDF showed the worst performance.

To ensure that the value obtained is consistent, standard deviation of each algorithm and turnaround time of each set of experiments were computed based on 32, 64 and 128 processors. Table 4.3, shows the standard deviation of the results graphed in Figure 4.3.

Table 4.3: Standard Deviation

Standard Deviation	FCFS	RR	EDF	EDFRR	EDFRRGCD	EDFRRLCM	LST	LSTRR	LSTRRGCD	LSTRRLCM	LSTRF	LSTRFRR	LSTRFRRGCD	LSTRFRRLCM
32	5.04E+10	5.04E+10	5.11E+10	5.05E+10	5.05E+10	5.05E+10	5.04E+10	5.05E+10	5.05E+10	5.05E+10	5.04E+10	5.05E+10	5.05E+10	5.05E+10
64	7.21E+10	7.21E+10	7.22E+10	7.22E+10	7.22E+10	7.22E+10	7.21E+10	7.22E+10	7.22E+10	7.22E+10	7.21E+10	7.22E+10	7.22E+10	9.13E+10
128	1.03E+11	1.03E+11	1.03E+11	1.03E+11	1.03E+11	1.03E+11	1.03E+11	1.03E+11	1.03E+11	1.03E+11	1.03E+11	1.03E+11	1.03E+11	1.03E+11

4.2.2 AuverGrid Traces

The experiment was carried out using the whole traces file workload.

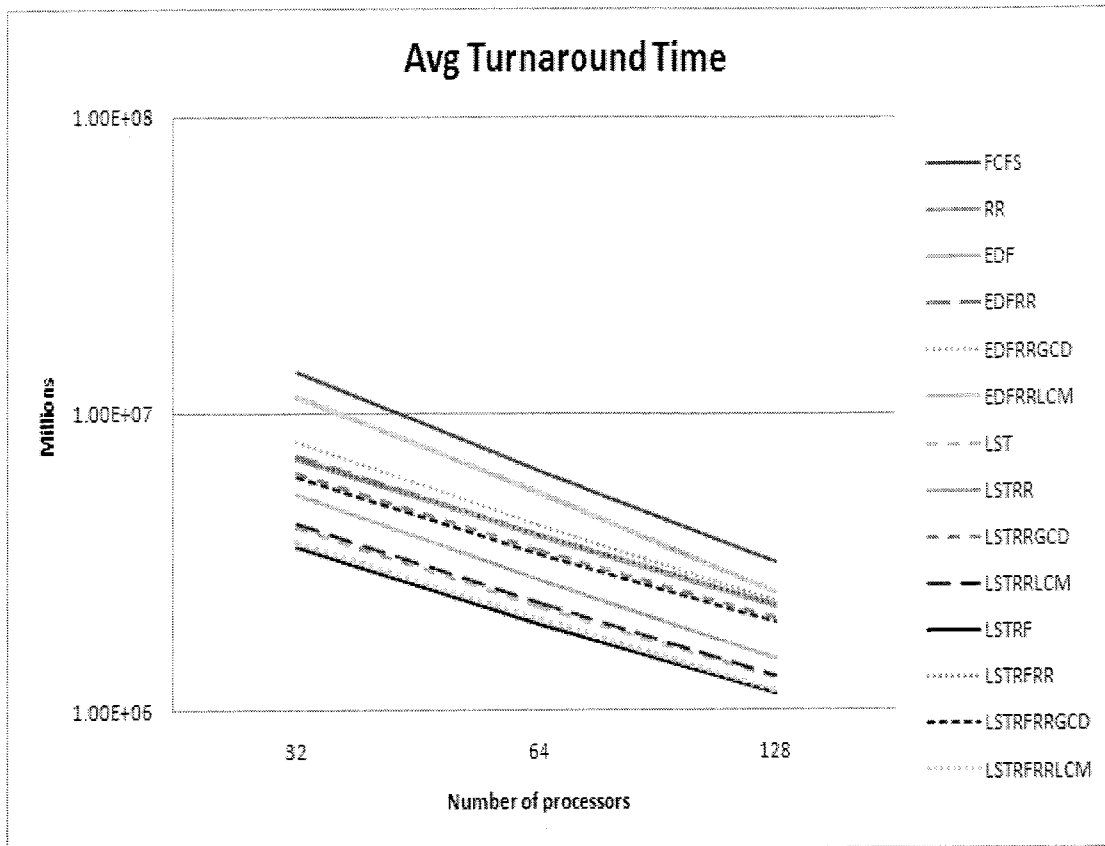


Figure 4.4: Average Turnaround Time

This experiment was carried out by varying the number of processors from 32, 64 and 128 numbers of processors. Based on the observation of Figure 4.4, it is clear that LSTRF, LST, RR, LSTRR, LSTRRGCD, LSTRRLCM, EDF, EDFRR, EDFRRGCD, EDFRRLCM, LSTRFRR, LSTRFRGCD, LSTRFRRLCM and FCFS are smooth and steady from 32 to 64 and 128 processors. Results showed that LSTRFRRLCM and LSTRF have the best performance, while FCFS and EDFRRLCM showed the worst performance.

To ensure that the value obtained is consistent, standard deviation of each algorithm and turnaround time of each set of experiments were computed based on 32, 64 and 128 processors. Table 4.4 shows the standard deviation of Figure 4.4 results.

Table 4.4: Standard Deviation

Standard Deviation														
	FCFS	RR	EDF	EDFRR	EDFRRGCD	EDFRRLCM	LST	LSTRR	LSTRRGCD	LSTRRLCM	LSTRF	LSTRFRR	LSTRFRRGCD	LSTRFRRLCM
32	6.03E+08	3.08E+08	2.34E+08	3.14E+08	3.49E+08	5.03E+08	1.81E+08	3.13E+08	2.76E+08	1.87E+08	1.55E+08	3.13E+08	2.88E+08	1.62E+08
64	4.03E+08	2.40E+08	1.74E+08	2.48E+08	2.63E+08	3.40E+08	1.40E+08	2.45E+08	2.18E+08	1.44E+08	1.23E+08	2.45E+08	2.13E+08	1.27E+08
128	2.84E+08	1.99E+08	1.33E+08	2.03E+08	2.05E+08	2.25E+08	1.15E+08	2.03E+08	1.83E+08	1.17E+08	1.03E+08	2.03E+08	1.79E+08	1.05E+08

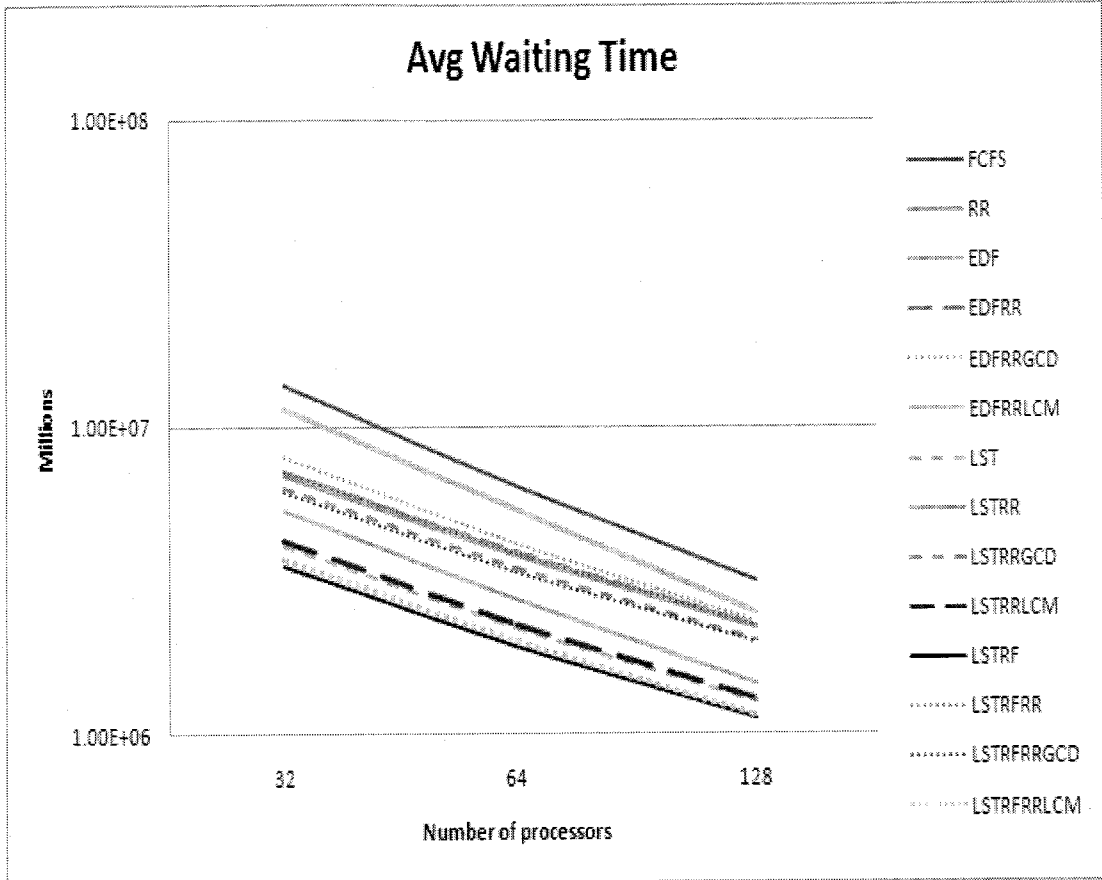


Figure 4.5: Average Waiting Time

It can be observed from Figure 4.5 that LSTRF, LST, RR, LSTRR, LSTRRGCD, LSTRRLCM, EDF, EDFRR, EDFRRGCD, EDFRRLCM, LSTRFRR, LSTRFRRGCD, LSTRFRRLCM and FCFS are smooth and steady from 32 to 64 and 128 processors. Results showed that LSTRFRRLCM and LSTRF have the best performance and EDFRRLCM showed the worst performance.

To ensure that the value obtained is consistent, standard deviation of each

algorithms turnaround time of each set of experiments, based on 32, 64 and 128 processors were computed. Table 4.5, showed the standard deviation of Figure 4.5 results.

Table 4.5: Standard Deviation

Standard Deviation	FCFS	RR	EDF	EDFRR	EDFRRGCD	EDFRRLCM	LST	LSTRR	LSTRRGCD	LSTRRLCM	LSTRF	LSTRFRR	LSTRFRRGCD	LSTRFRRLCM
32	6.02E+08	3.05E+08	2.34E+08	3.13E+08	3.49E+08	5.02E+08	1.80E+08	3.12E+08	2.76E+08	1.88E+08	1.58E+08	3.13E+08	2.68E+08	1.51E+08
64	4.02E+08	2.39E+08	1.74E+08	2.45E+08	2.62E+08	3.39E+08	1.39E+08	2.44E+08	2.17E+08	1.43E+08	1.22E+08	2.45E+08	2.12E+08	1.26E+08
128	2.83E+08	1.98E+08	1.31E+08	2.02E+08	2.02E+08	2.24E+08	1.14E+08	2.01E+08	1.81E+08	1.18E+08	1.01E+08	2.02E+08	1.77E+08	1.04E+08

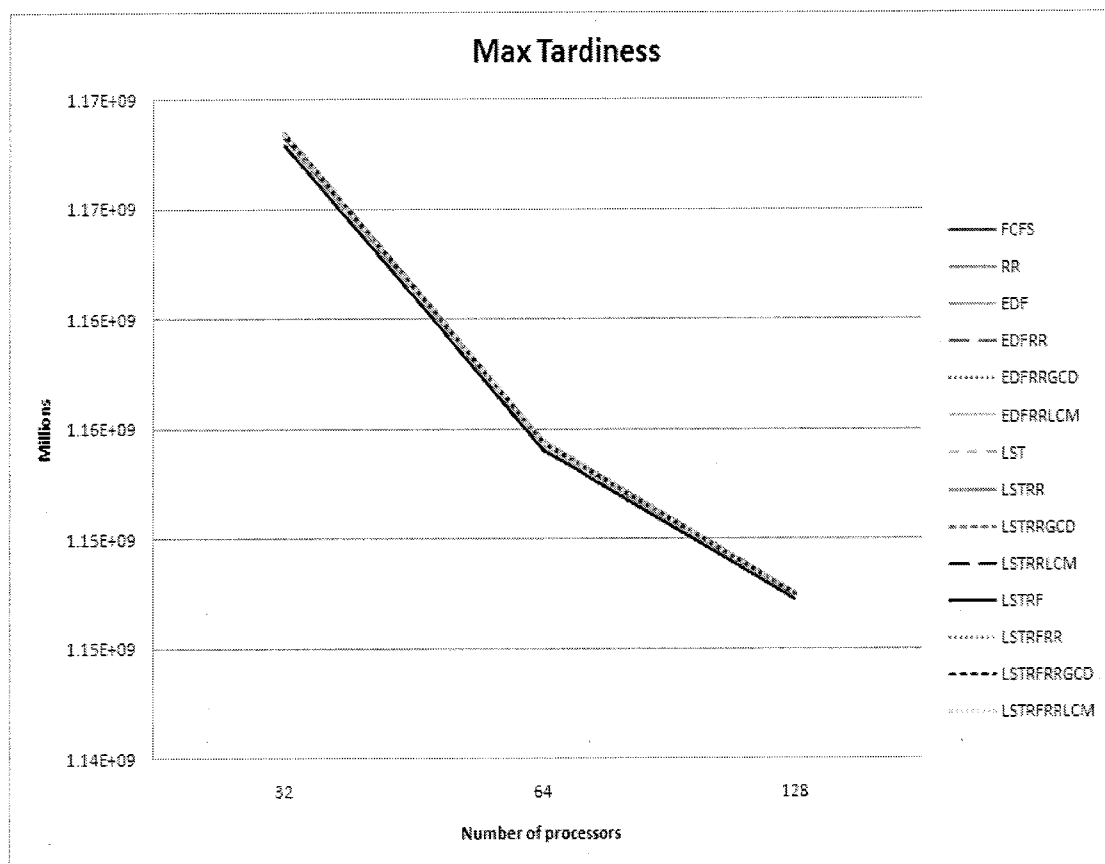


Figure 4.6: Maximum Tardiness

Figure 4.6 shows that maximum tardiness is not fixed it vary on the workload. However, it can be observed from that LSTRF, LST, RR, LSTRR, LSTRRGCD, LSTRRLCM, EDFRR, EDFRRGCD, EDFRRLCM, LSTRFRR, LSTRFRRGCD, LSTRFRRLCM and FCFS are smooth and steady as well as overlap one another from

32 to 64 and 128 processors. However, there is a slight difference between the algorithms from. Results showed that LSTRF and LSTRFRRLCM have the best performance, while EDFRRLCM and FCFS showed the worst performance.

To ensure that the value obtained is consistent, standard deviation of each algorithm and turnaround time of each set of experiments were computed based on 32, 64 and 128 processors. Table 4.6, showed the standard deviation of Figure 4.6 results.

Table 4.6: Standard Deviation

Standard Deviation														
	FCFS	RR	EDF	EDFRR	EDFRRGCD	EDFRRLCM	LST	LSTRR	LSTRRGCD	LSTRRLCM	LSTRF	LSTRFRR	LSTRFRRGCD	LSTRFRRLCM
32	5.12E+10	5.12E+10	5.12E+10	5.12E+10	5.12E+10	5.12E+10	5.12E+10	5.12E+10	5.12E+10	5.12E+10	5.12E+10	5.12E+10	5.12E+10	5.12E+10
64	7.27E+10	7.27E+10	7.27E+10	7.27E+10	7.27E+10	7.27E+10	7.27E+10	7.27E+10	7.27E+10	7.27E+10	7.27E+10	7.27E+10	7.27E+10	7.27E+10
128	1.03E+11	1.03E+11	1.03E+11	1.03E+11	1.03E+11	1.03E+11	1.03E+11	1.03E+11	1.03E+11	1.03E+11	1.03E+11	1.03E+11	1.03E+11	1.03E+11

4.2.3 Grid5000 Traces

The experiment was carried out using the whole traces file workload.

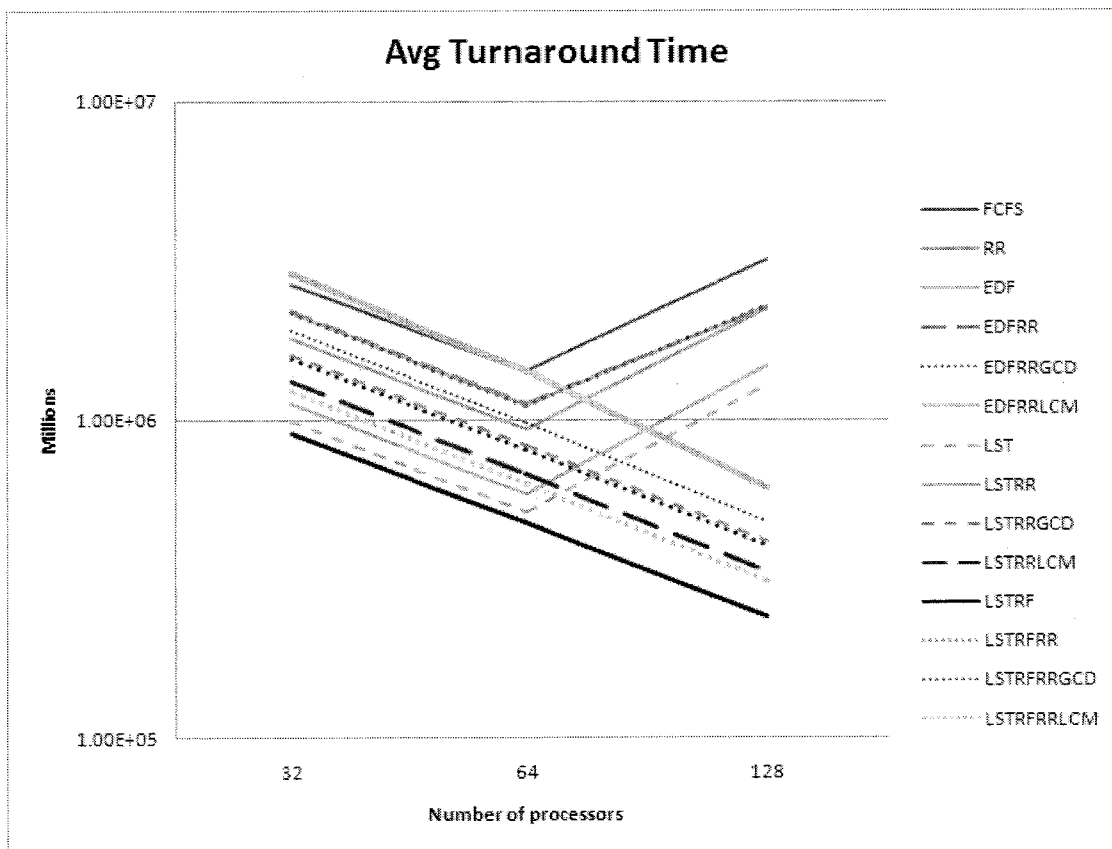


Figure 4.7: Average Turnaround Time

This experiment was carried out by varying the number of processors from 32, 64 and 128 numbers of processors. Based on the observations shown Figure 4.7, it's clear that LSTRF, LST, LSTRRGCD, LSTRRLCM, EDFRRGCD, EDFRRLCM, LSTRFRRGCD, and LSTRFRRLCM are smooth and steady fall from 32 to 64 and 128 processors. Meanwhile, LSTRR, EDFRR, LSTRFRR, EDF, RR and FCFS have sharp fall from 32 to 64 processors and a sharp rise from 64 to 128 processors, possible reason may be heavy workload and increase in number of processors. LSTRR, EDFRR, LSTRFRR are overlapping one another. But there is a slight difference between the algorithms. Results showed that LSTRF and LSTRFRRLCM have the best performance, while FCFS, showed the worst performance.

To ensure that the value obtained is consistent, standard deviation of each algorithms turnaround time of each set of experiments, based on 32, 64 and 128 processors were computed. Table 4.7, showed the standard deviation of Figure 4.7 results.

Table 4.7: Standard Deviation

Standard Deviation	FCFS	RR	EDF	EDFRR	EDFRRGCD	EDFRRLCM	LST	LSTRR	LSTRRGCD	LSTRRLCM	LSTRF	LSTRFRR	LSTRFRRGCD	LSTRFRRLCM
32	1.17E+08	7.95E+07	4.93E+07	9.60E+07	8.40E+07	1.28E+08	4.32E+07	9.51E+07	7.04E+07	5.78E+07	3.97E+07	9.53E+07	6.88E+07	5.37E+07
64	8.97E+07	5.90E+07	3.70E+07	7.07E+07	6.14E+07	8.98E+07	3.28E+07	6.99E+07	5.23E+07	4.25E+07	3.00E+07	7.01E+07	5.11E+07	3.98E+07
128	2.84E+08	1.99E+08	1.33E+08	2.03E+08	4.32E+07	5.52E+07	1.16E+08	2.03E+08	3.73E+07	3.00E+07	2.16E+07	2.03E+08	3.64E+07	2.80E+07

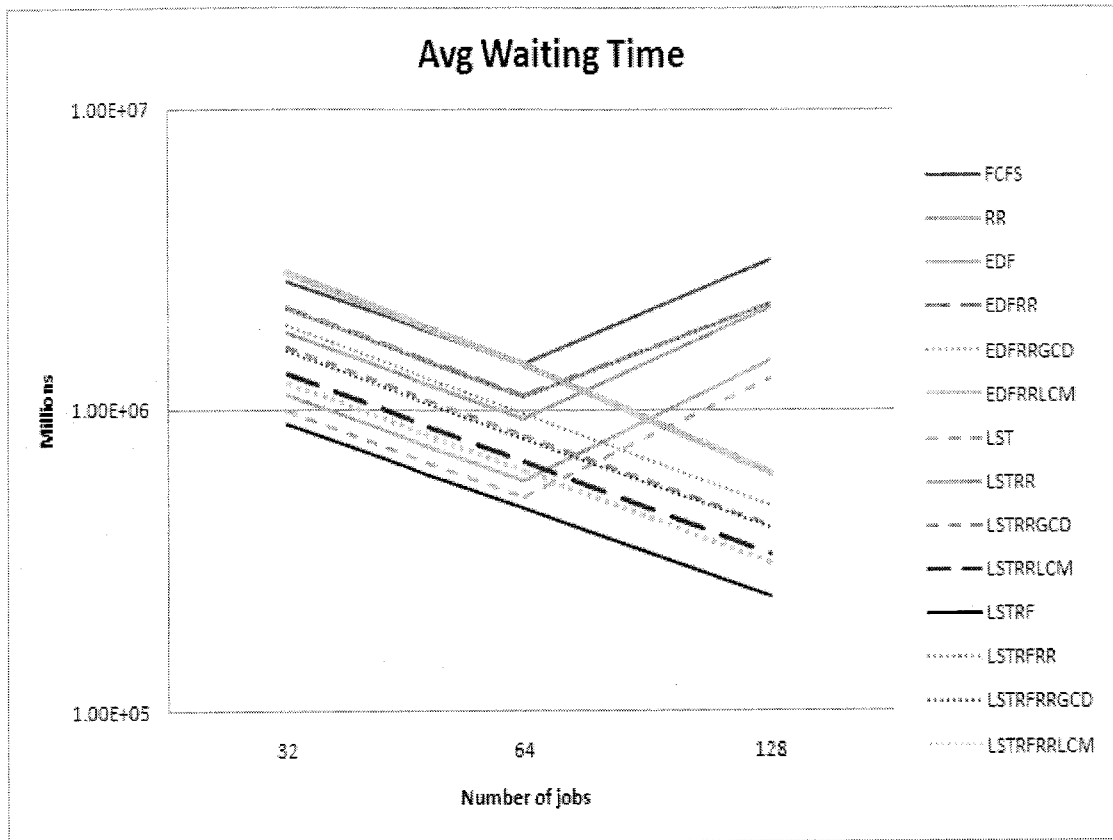


Figure 4.8: Average Waiting Time

It can be observed from Figure 4.8 that that LSTRF, LST, LSTRRGCD, LSTRRLCM, EDFRRGCD, EDFRRLCM, LSTRFRRGCD, and LSTRFRRLCM are smooth and steady from 32 to 64 and 128 processors. Meanwhile, LSTRR, EDFRR, LSTRFRR, EDF, RR and FCFS have sharp fall from 32 to 64 processors and a sharp rise from 64 to 128 processors. LSTRR, EDFRR, LSTRFRR are overlapping one another. But there is a slight difference between the algorithms from. Results showed that LSTRF and LSTRFRRLCM have the best performance, while FCFS, showed the worst performance.

To ensure that the value obtained is consistent, standard deviation of each algorithm and turnaround time of each set of experiments were computed based on 32, 64 and 128 processors. Table 4.8, showed the standard deviation of Figure 4.8 results.

Table 4.8: Standard Deviation

Standard Deviation	FCFS	RR	EDF	EDFRR	EDFRRGCD	EDFRRLCM	LST	LSTRR	LSTRRGCD	LSTRRLCM	LSTRF	LSTRFRR	LSTRFRRGCD	LSTRFRRLCM
32	1.17E+08	7.94E+07	4.92E+07	9.58E+07	8.39E+07	1.26E+08	4.31E+07	9.49E+07	7.03E+07	5.77E+07	3.95E+07	9.52E+07	6.88E+07	5.36E+07
64	8.95E+07	5.88E+07	3.68E+07	7.06E+07	6.12E+07	8.94E+07	3.23E+07	6.97E+07	5.21E+07	4.24E+07	2.98E+07	6.99E+07	5.09E+07	3.96E+07
128	2.83E+08	1.98E+08	1.03E+11	2.02E+08	4.29E+07	5.50E+07	1.14E+08	2.01E+08	3.78E+07	2.97E+07	2.14E+07	2.02E+08	3.62E+07	2.78E+07

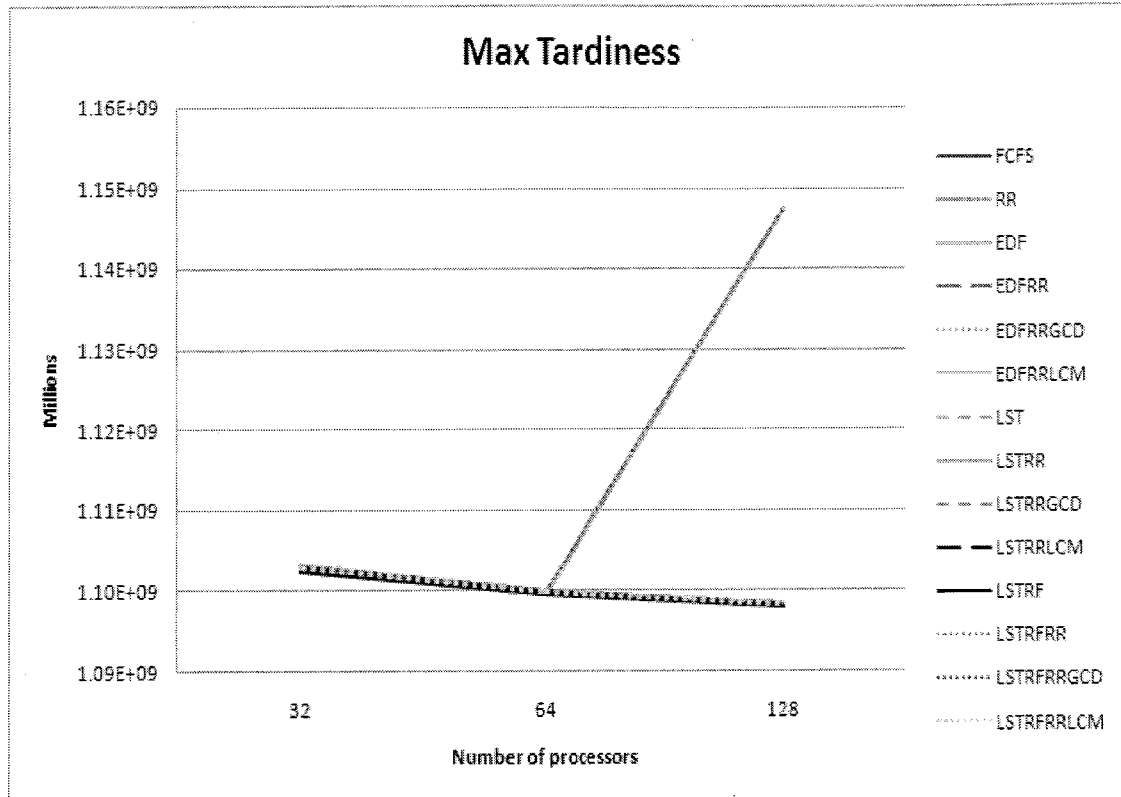


Figure 4.9: Maximum Tardiness

Figure 4.9 shows that maximum tardiness is not fixed; it varies with the workload. However, it can be observed that LSTRF, LST, RR, LSTRR, LSTRRGCD, LSTRRLCM, EDFRR, EDFRRGCD, EDFRRLCM, LSTRFRR, LSTRFRRGCD, LSTRFRRLCM and FCFS are smooth and steady as well as overlap one another from 32 to 64 and 128 processors. Nevertheless, there is a slight difference between the algorithms. Meanwhile, LSTRR, EDFRR, LSTRFRR, EDF, RR and FCFS have sharp fall from 32 to 64 processors and a sharp rise from 64 to 128 processors. Results showed that LSTRF and LSTRFRRLCM have the best performance, while EDFRR and EDF showed the worst performance.

To ensure that the value obtained is consistent, standard deviation of each algorithm and turnaround time of each set of experiments were computed based on 32, 64 and 128 processors. Table 4.9, showed the standard deviation of Figure 4.9 results.

Table 4.9: Standard Deviation

Standard Deviation	FCFS	RR	EDF	EDFRR	EDFRRGCD	EDFRRLCM	LST	LSTRR	LSTRRGCD	LSTRRLCM	LSTRF	LSTRFRR	LSTRFRRGCD	LSTRFRRLCM
32	4.03E+10	4.03E+10	4.03E+10	4.03E+10	4.03E+10	4.03E+10	4.03E+10	4.03E+10	4.03E+10	4.03E+10	4.03E+10	4.03E+10	4.03E+10	4.03E+10
64	6.93E+10	6.93E+10	6.93E+10	6.93E+10	6.93E+10	6.93E+10	6.93E+10	6.93E+10	6.93E+10	6.93E+10	6.93E+10	6.93E+10	6.93E+10	6.93E+10
128	1.03E+11	1.03E+11	1.03E+11	1.03E+11	9.06E+10	9.06E+10	1.03E+11	1.03E+11	9.06E+10	9.06E+10	9.06E+10	1.03E+11	9.06E+10	9.06E+10

4.2.4 LCG Traces

The experiment was carried out using the whole traces file workload.

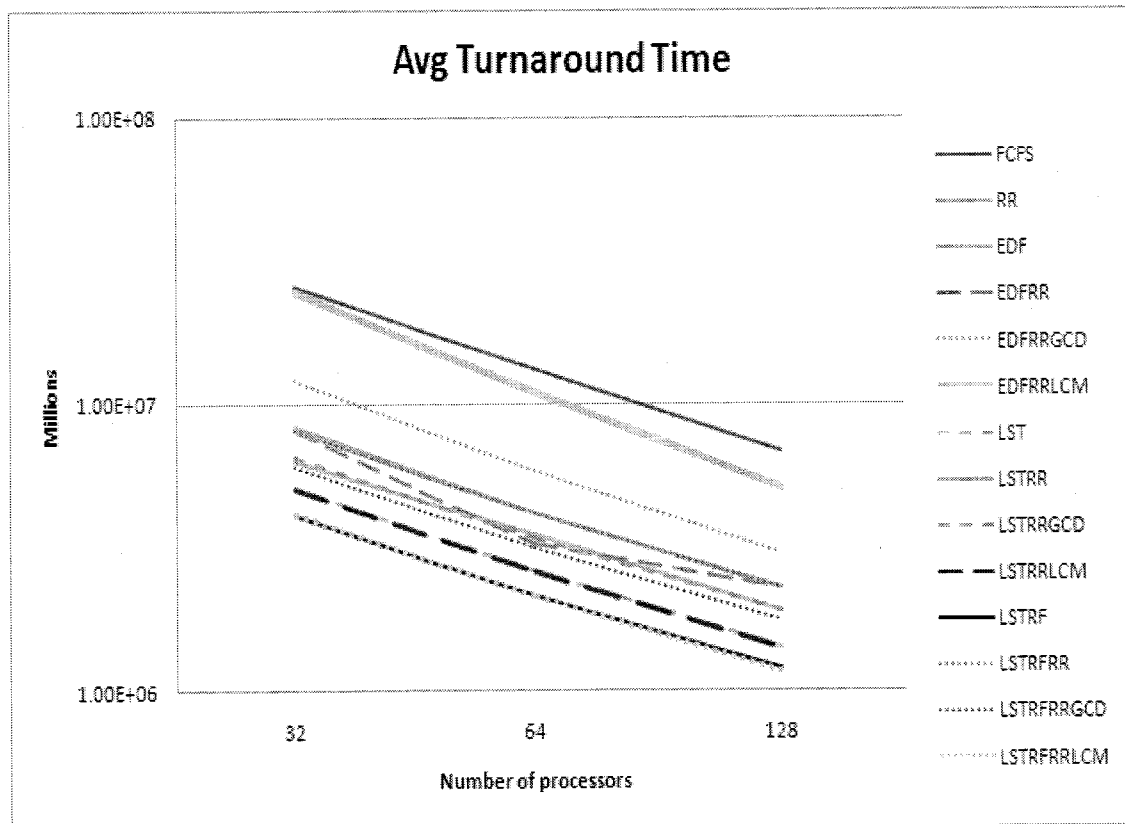


Figure 4.10: Average Turnaround Time

This experiment was carried out by varying the number of processors from 32, 64 and 128 numbers of processors. Based on the observation of Figure 4.4, it's clear

that LSTRF, LST, RR, LSTRR, LSTRRGCD, LSTRRLCM, EDF, EDFRR, EDFRRGCD, EDFRRLCM, LSTRFRR, LSTRFRRGCD, LSTRFRRLCM and FCFS are smooth and steady from 32 to 64 and 128 processors. LSTRR, EDFRR, LSTRFRR, LST, LSTRRLCM, LSTRF and LSTRFRRLCM are overlapping one another. But there is a slight difference between the algorithms. But EDFRR showed a slight fall from 32 to 64 processors and a slight rise from 64 to 128 processors. Results showed that LSTRFRRLCM and LSTRF have the best performance, while FCFS and EDFRRLCM showed the worst performance.

To ensure that the value obtained is consistent, standard deviation of each algorithm and turnaround time of each set of experiments were computed based on 32, 64 and 128 processors. Table 4.10, showed the standard deviation of Figure 4.10 results.

Table 4.10: Standard Deviation

Standard Deviation	FCFS	RR	EDF	EDFRR	EDFRRGCD	EDFRRLCM	LST	LSTRR	LSTRRGCD	LSTRRLCM	LSTRF	LSTRFRR	LSTRFRRGCD	LSTRFRRLCM
32	1.13E+09	3.68E+08	2.75E+08	3.55E+08	5.26E+08	1.07E+09	2.22E+08	3.53E+08	2.85E+08	2.22E+08	1.81E+08	3.53E+08	2.64E+08	1.81E+08
64	8.39E+08	2.63E+08	2.22E+08	2.04E+08	3.68E+08	6.93E+08	1.64E+08	2.82E+08	2.13E+08	1.63E+08	1.36E+08	2.82E+08	1.99E+08	1.36E+08
128	5.14E+08	2.05E+08	1.74E+08	2.06E+08	2.72E+08	4.57E+08	1.27E+08	2.05E+08	1.69E+08	1.28E+08	1.07E+08	2.05E+08	1.58E+08	1.06E+08

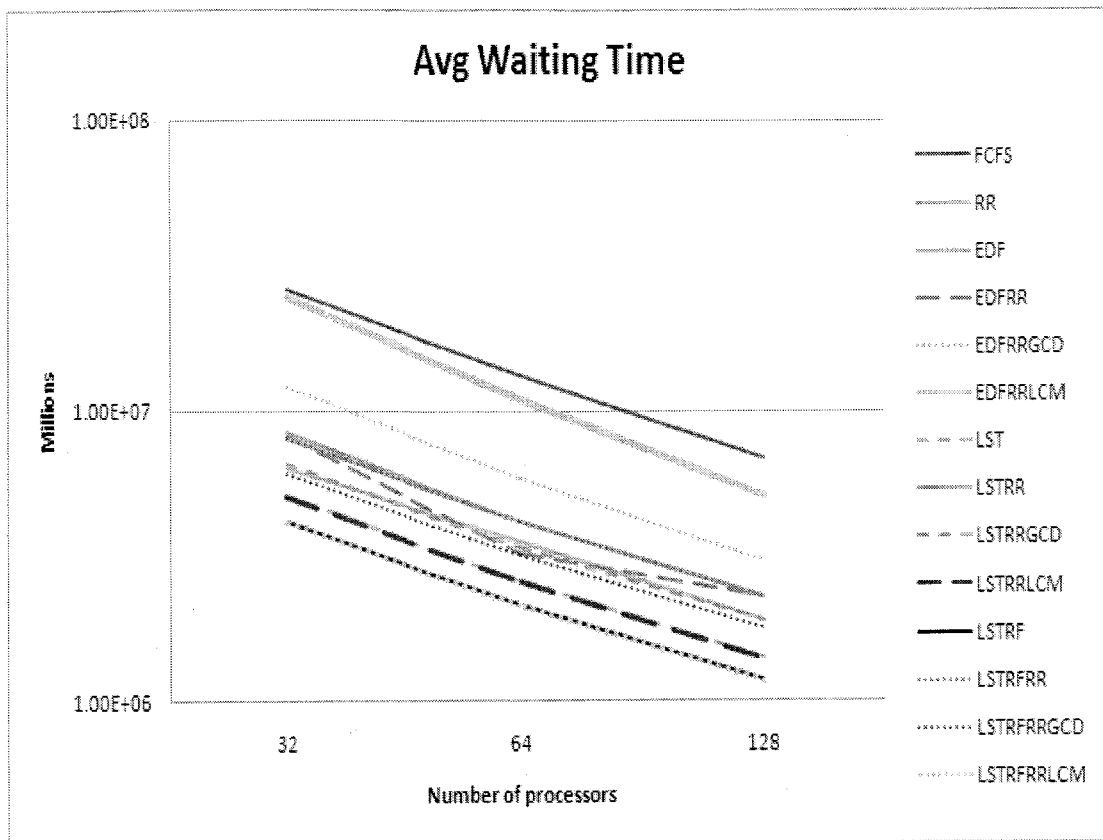


Figure 4.11: Average Waiting Time

It can be observed from Figure 4.11 that LSTRF, LST, RR, LSTRR, LSTRRGCD, LSTRRLCM, EDF, EDFRR, EDFRRGCD, EDFRRLCM, LSTRFRR, LSTRFRRGCD, LSTRFRLCM and FCFS are smooth and steady from 32 to 64 and 128 processors. LSTRR, EDFRR, LSTRFRR, LST, LSTRRLCM, LSTRF and LSTRFRLCM are overlapping one another. But there is a slight difference between the algorithms. But EDFRR showed a slight fall from 32 to 64 processors and a slight rise from 64 to 128 processors. Results showed that LSTRFRLCM and LSTRF have the best performance, while FCFS and EDFRRLCM showed the worst performance.

To ensure that the value obtained is consistent, standard deviation of each algorithms turnaround time of each set of experiments, based on 32, 64 and 128 processors were computed. Table 4.11, showed the standard deviation of Figure 4.11 results.

Table 4.11: Standard Deviation

Standard Deviation														
	FCFS	RR	EDF	EDFRR	EDFRRGCD	EDFRRLCM	LST	LSTRR	LSTRRGCD	LSTRRLCM	LSTRF	LSTRFRR	LSTRFRRGCD	LSTRFRRLCM
32	1.13E+09	3.68E+08	2.75E+08	3.54E+08	5.28E+08	1.07E+09	2.22E+08	3.52E+08	2.85E+08	2.22E+08	1.80E+08	3.52E+08	2.64E+08	1.80E+08
64	8.39E+08	2.62E+08	2.21E+08	2.04E+08	3.67E+08	6.92E+08	1.63E+08	2.62E+08	2.13E+08	1.63E+08	1.34E+08	2.62E+08	1.98E+08	1.34E+08
128	6.14E+08	2.04E+08	1.71E+08	2.05E+08	2.71E+08	4.58E+08	1.27E+08	2.04E+08	1.68E+08	1.26E+08	1.09E+08	2.04E+08	1.58E+08	1.04E+08

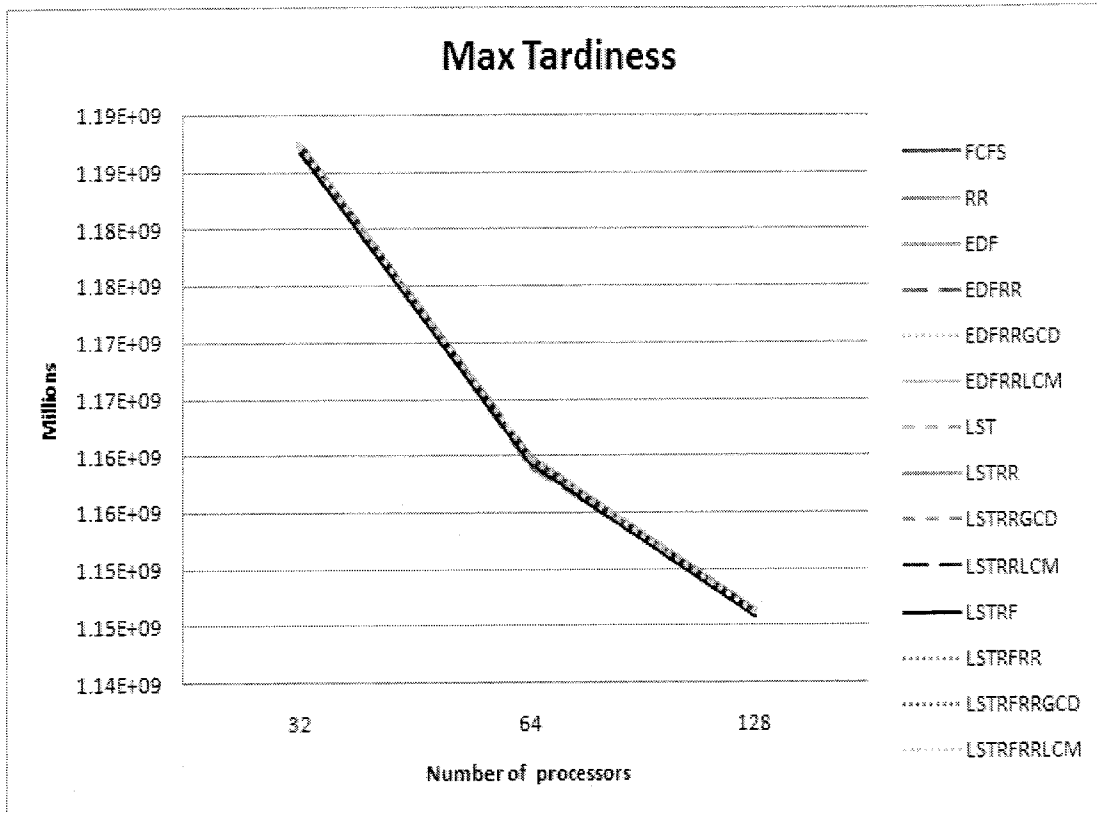


Figure 4.12: Maximum Tardiness

Figure 4.12 shows that maximum tardiness is not fixed it varies on the workload. However, it can be observed that LSTRF, LST, RR, LSTRR, LSTRRGCD, LSTRRLCM, EDFRR, EDFRRGCD, EDFRRLCM, LSTRFRR, LSTRFRRGCD, LSTRFRRLCM and FCFS showed a sharp fall from 32 to 64 processors and from 64 to 128 processors as well as overlap one another. Results showed that LSTRF and LSTRFRRLCM have the best performance. While EDF and FCFS showed the worst performance.

To ensure that the value obtained is consistent, standard deviation of each

algorithms turnaround time of each set of experiments, based on 32, 64 and 128 processors were computed. Table 4.12, showed the standard deviation of Figure 4.12 results.

Table 4.12: Standard Deviation

Standard Deviation		FCFS	RR	EDF	EDFRR	EDFRRGCD	EDFRRLCM	LST	LSTRR	LSTRRGCD	LSTRRLCM	LSTRF	LSTRFRR	LSTRFRRGCD	LSTRFRRLCM
32	5.20E+10	5.20E+10	5.20E+10	5.20E+10	5.20E+10	5.20E+10	5.20E+10	5.20E+10	5.20E+10	5.20E+10	5.20E+10	5.20E+10	5.20E+10	5.20E+10	5.20E+10
64	7.31E+10	7.30E+10	7.30E+10	7.30E+10	7.31E+10	7.31E+10	7.31E+10	7.30E+10	7.31E+10	7.31E+10	7.31E+10	7.30E+10	7.31E+10	7.31E+10	7.31E+10
128	1.03E+11	2.04E+08	1.03E+11	1.03E+11	1.03E+11	1.03E+11	1.03E+11	1.03E+11	1.03E+11	1.03E+11	1.03E+11	1.03E+11	1.03E+11	1.03E+11	1.03E+11

4.2.5 NorduGrid Traces

The experiment was carried out using the whole traces file workload.

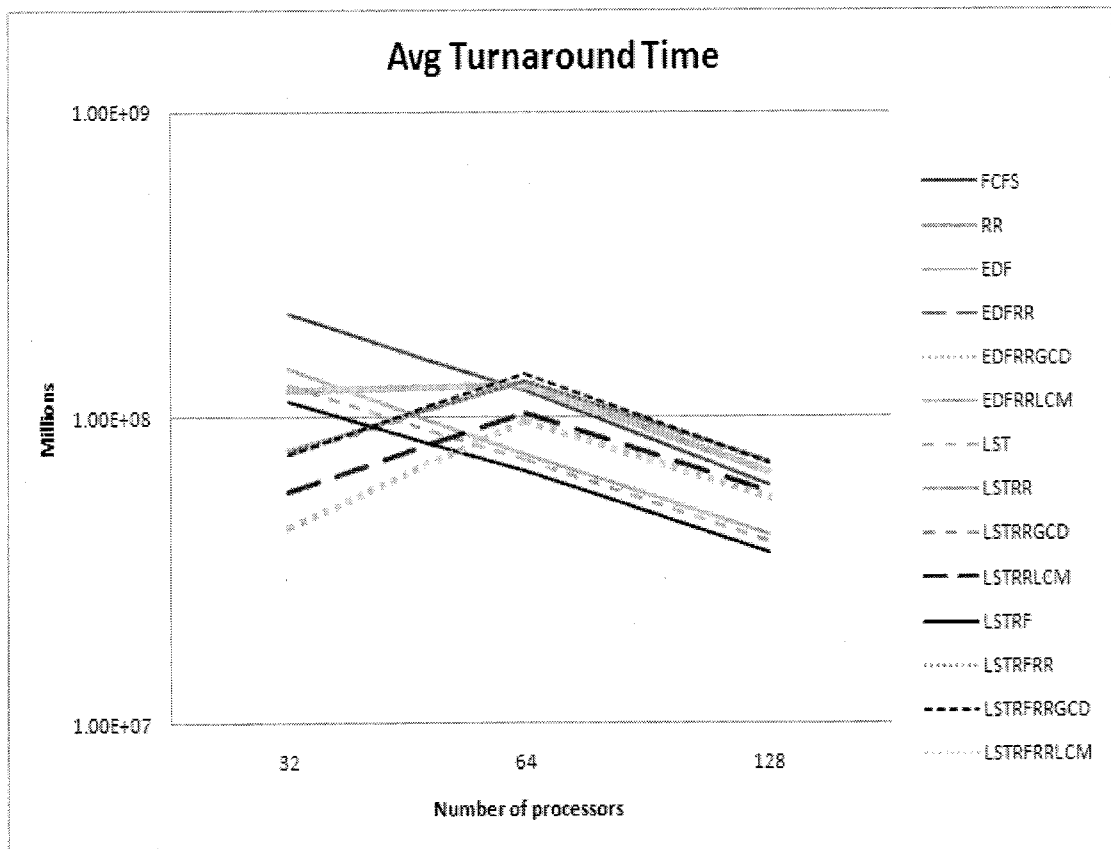


Figure 4.13: Average Turnaround Time

This experiment was carried out by varying the number of processors from 32, 64 and 128 numbers of processors. Based on the observation of Figure 4.13, it's

clear that LSTRF, LST, LSTRRGCD, LSTRRLCM, EDFRRGCD, EDFRRLCM, LSTRFRRGCD, and LSTRFRRLCM are smooth and steady from 32 to 64 and 128 processors. Meanwhile, LSTRR, EDFRR, LSTRFRR, EDF, RR and FCFS have a sharp rise from 32 to 64 processors and a sharp fall from 64 to 128 processors. LSTRR, EDFRR, LSTRFRR are overlapping one another. But there is a slight difference between the algorithms. Results showed that LSTRF and LST have the best performance, while FCFS, showed the worst performance.

To ensure that the value obtained is consistent, standard deviation of each algorithms turnaround time of each set of experiments, based on 32, 64 and 128 processors were computed. Table 4.13, showed the standard deviation of Figure 4.13 results.

Table 4.13: Standard Deviation

Standard Deviation	FCFS	RR	EDF	EDFRR	EDFRRGCD	EDFRRLCM	LST	LSTRR	LSTRRGCD	LSTRRLCM	LSTRF	LSTRFRR	LSTRFRRGCD	LSTRFRRLCM
32	9.61E+09	3.39E+09	6.33E+09	3.42E+09	3.46E+09	5.40E+09	5.57E+09	3.42E+09	3.34E+09	2.51E+09	4.93E+09	3.42E+09	3.33E+09	1.93E+09
64	7.70E+09	8.27E+09	4.75E+09	8.28E+09	8.29E+09	8.00E+09	4.61E+09	8.30E+09	8.22E+09	6.48E+09	4.20E+09	8.30E+09	8.13E+09	6.09E+09
128	5.37E+09	6.39E+09	3.70E+09	6.41E+09	6.39E+09	5.95E+09	3.52E+09	6.41E+09	6.36E+09	5.13E+09	3.25E+09	6.41E+09	6.34E+09	4.87E+09

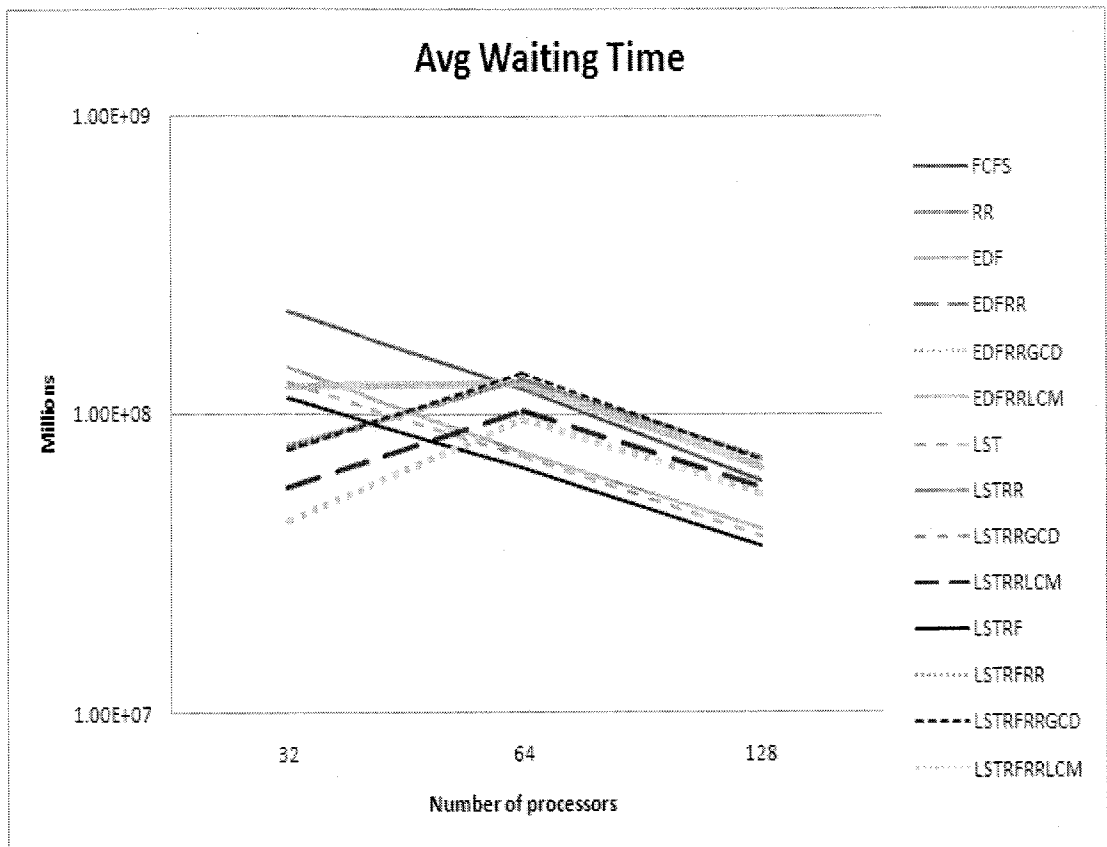


Figure 4.14: Average Waiting Time

It can be observed from Figure 4.14 that LSTRF, LST, LSTRRGCD, LSTRRLCM, EDFRRGCD, EDFRRLCM, LSTRFRRGCD, and LSTRFRRLCM are smooth and steady from 32 to 64 and 128 processors. Meanwhile, LSTRR, EDFRR, LSTRFRR, EDF, RR and FCFS have a sharp rise from 32 to 64 processors and a sharp fall from 64 to 128 processors. LSTRR, EDFRR, LSTRFRR are overlapping one another. But there is a slight difference between the algorithms. Results showed that LSTRF and LST have the best performance, while FCFS, showed the worst performance.

To ensure that the value obtained is consistent, standard deviation of each algorithms turnaround time of each set of experiments, based on 32, 64 and 128 processors were computed. Table 4.14, showed the standard deviation of Figure 4.14 results.

Table 4.14: Standard Deviation

Standard Deviation	FCFS	RR	EDF	EDFRR	EDFRRGCD	EDFRRLCM	LST	LSTRR	LSTRRGCD	LSTRRLCM	LSTRF	LSTRFRR	LSTRFRRGCD	LSTRFRRLCM
32	9.60E+09	3.38E+09	6.32E+09	3.41E+09	3.45E+09	5.30E+09	5.58E+09	3.41E+09	3.33E+09	2.50E+09	4.92E+09	3.41E+09	3.32E+09	1.92E+09
64	7.69E+09	8.25E+09	7.51E+07	8.28E+09	8.27E+09	7.98E+09	4.59E+09	8.28E+09	8.21E+09	6.47E+09	4.18E+09	8.28E+09	8.11E+09	6.07E+09
128	5.35E+09	6.37E+09	3.68E+09	6.39E+09	6.37E+09	5.99E+09	3.50E+09	6.39E+09	6.34E+09	5.10E+09	3.23E+09	6.39E+09	6.32E+09	4.85E+09

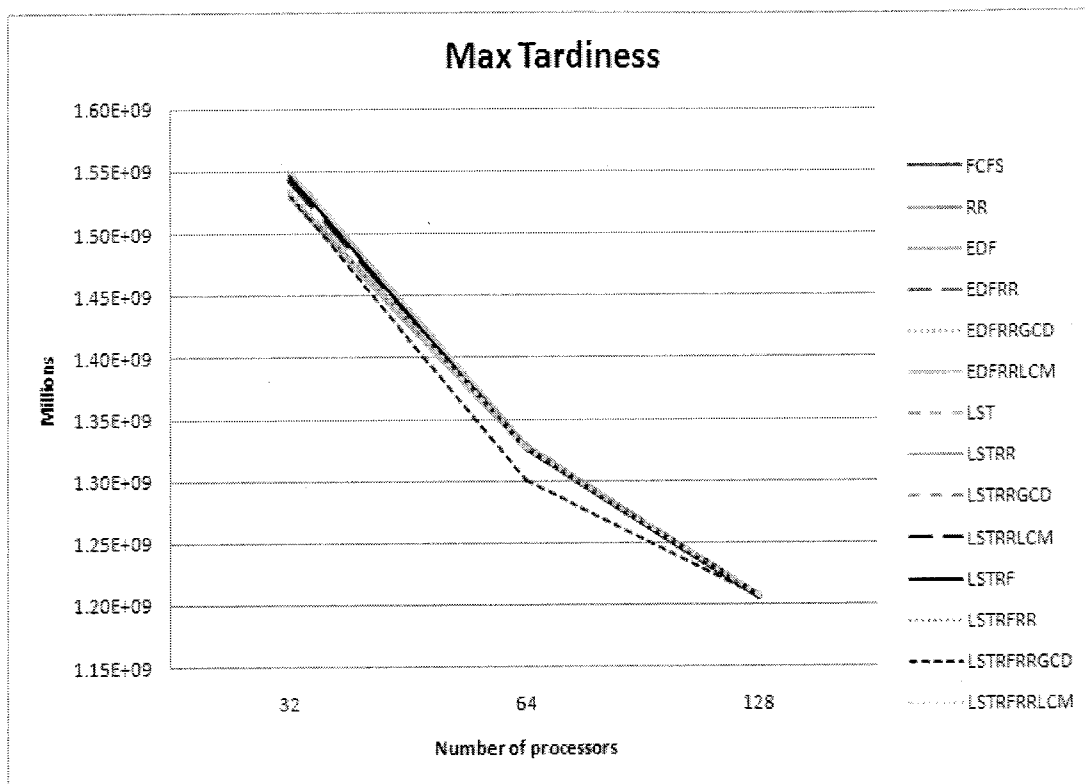


Figure 4.15: Maximum Tardiness

Figure 4.15 shows that maximum tardiness is not fixed it vary on the workload. However, it can be observed from the graph point of view that that LSTRF, LST, RR, LSTRR, LSTRRGCD, LSTRRLCM, EDFRR, EDFRRGCD, EDFRRLCM, LSTRFRR, LSTRFRRLCM and FCFS are sharp fall as well as overlap one another from 32 to 64 processors and from 64 to 128 processors. But there is a slight difference between the algorithms. Meanwhile, LSTRFRRGCD showed a slight sharp fall from 32 to 64 processors and from 64 to 128 processors. Results showed that LSTRFRRGCD and LSTRF have the best performance, while EDFRRLCM and LST showed the worst performance.

To ensure that the value obtained is consistent, standard deviation of each algorithms turnaround time of each set of experiments, based on 32, 64 and 128 processors were computed. Table 4.15, showed the standard deviation of Figure 4.15 results.

Table 4.15: Standard Deviation

Standard Deviation	FCFS	RR	EDF	EDFRR	EDFRRGCD	EDFRRLCM	LST	LSTRR	LSTRRGCD	LSTRRLCM	LSTRF	LSTRFRR	LSTRFRRGCD	LSTRFRRLCM
32	6.79E+10	6.70E+10	6.77E+10	6.71E+10	6.71E+10	6.79E+10	6.77E+10	6.71E+10	6.71E+10	6.76E+10	6.77E+10	6.71E+10	6.71E+10	6.72E+10
64	8.36E+10	8.35E+10	8.34E+10	8.36E+10	8.36E+10	8.36E+10	8.34E+10	8.36E+10	8.36E+10	8.36E+10	8.34E+10	8.36E+10	7.66E+10	8.36E+10
128	1.08E+11	1.08E+11	1.08E+11	1.08E+11	1.08E+11	1.08E+11	1.08E+11	1.08E+11	1.08E+11	1.08E+11	1.08E+11	1.08E+11	1.08E+11	1.08E+11

4.2.6 Sharcnet Traces

The experiment was carried out using the whole traces file workload.

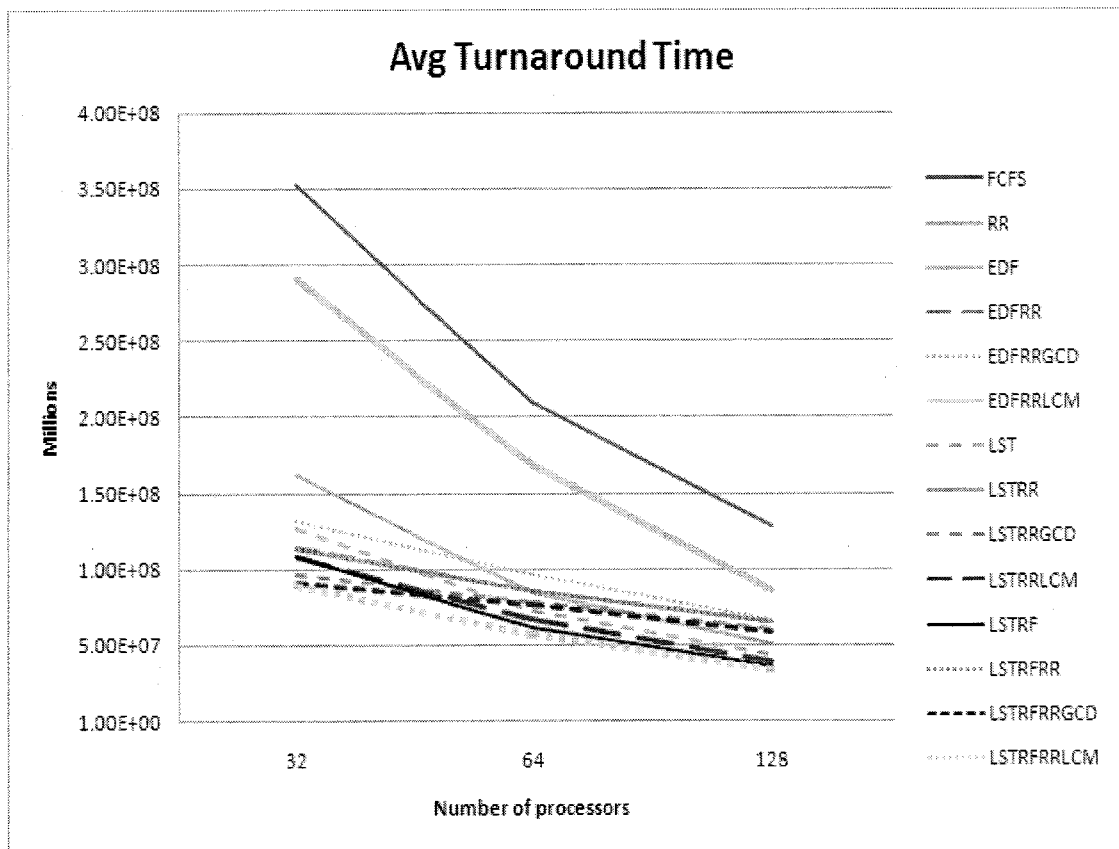


Figure 4.16: Average Turnaround Time

This experiment was carried out by varying the number of processors from 32, 64

and 128 numbers of processors. Based on the observation of Figure 4.15, it is clear that LSTRF, LST, RR, LSTRR, LSTRRLCM, EDFRR, EDFRRGCD, LSTRFRR and LSTRFRRLCM are smooth and steady from 32 to 64 and 128 processors. LSTRFRRGCD and LSTRRGCD showed a sharp and steady fall from 32 to 64 and 128 processors. EDF showed a sharp fall from 32 to 64 processors and a sharp fall from 64 to 128 processors. Meanwhile, FCFS and EDFRRLCM showed a sharp fall from 32 to 64 processors and 64 to 128 processors. LSTRR, EDFRR, LSTRFRR, LST, LSTRRLCM, LSTRF and LSTRFRRLCM are overlapping one another. But there is a slight difference between the algorithms. Results showed that LSTRFRRLCM and LSTRF have the best performance, while FCFS and EDFRRLCM showed the worst performance.

To ensure that the value obtained is consistent, standard deviation of each algorithms turnaround time of each set of experiments, based on 32, 64 and 128 processors were computed. Table 4.16, showed the standard deviation of Figure 4.16 results.

Table 4.16: Standard Deviation

Standard Deviation	FCFS	RR	EDF	EDFRR	EDFRRGCD	EDFRRLCM	LST	LSTRR	LSTRRGCD	LSTRRLCM	LSTRF	LSTRFRR	LSTRFRRGCD	LSTRFRRLCM
32	1.54E+10	4.97E+09	7.09E+09	5.00E+09	5.77E+09	1.28E+10	5.54E+09	4.99E+09	4.22E+09	4.75E+09	4.75E+09	4.99E+09	4.01E+09	3.92E+09
64	1.32E+10	5.40E+09	5.34E+09	5.42E+09	6.07E+09	1.09E+10	4.53E+09	5.42E+09	4.94E+09	4.24E+09	3.84E+09	5.42E+09	4.81E+09	3.59E+09
128	1.15E+10	5.78E+09	4.53E+09	5.80E+09	5.93E+09	7.64E+09	3.82E+09	5.79E+09	5.34E+09	3.49E+09	3.28E+09	5.79E+09	5.20E+09	2.99E+09

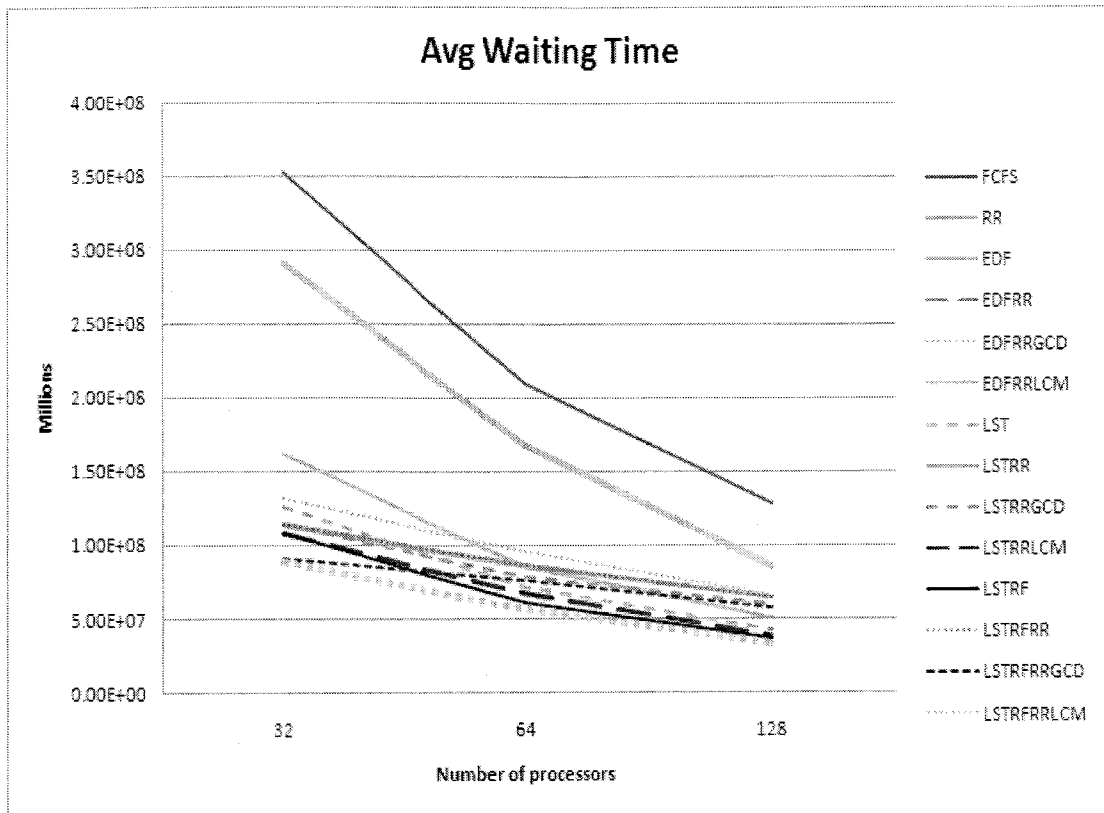


Figure 4.17: Average Waiting Time

It can be observed from Figure 4.17 that LSTRF, LST, RR, LSTRR, LSTRRLCM, EDFRR, EDFRRGCD, LSTRFRR and LSTRFRRLCM are smooth and steady from 32 to 64 and 128 processors. LSTRFRRGCD and LSTRRGCD showed a sharp and steady fall from 32 to 64 and 128 processors. EDF showed a sharp fall from 32 to 64 processors and a sharp fall from 64 to 128 processors. Meanwhile, FCFS and EDFRRLCM showed a sharp fall from 32 to 64 processors and 64 to 128 processors. LSTRR, EDFRR, LSTRFRR, LST, LSTRRLCM, LSTRF and LSTRFRRLCM are overlapping one another. But there is a slight difference between the algorithms. Results showed that LSTRFRRLCM and LSTRF have the best performance, while FCFS and EDFRRLCM showed the worst performance.

To ensure that the value obtained is consistent, standard deviation of each algorithms turnaround time of each set of experiments, based on 32, 64 and 128 processors were computed. Table 4.17, showed the standard deviation of Figure 4.17 results.

Table 4.17: Standard Deviation

Standard Deviation	FCFS	RR	EDF	EDFRR	EDFRRGCD	EDFRRLCM	LST	LSTRR	LSTRRGCD	LSTRRLCM	LSTRF	LSTRFRR	LSTRFRRGCD	LSTRFRRLCM
32	1.54E+10	4.97E+09	7.09E+09	5.00E+09	5.77E+09	1.26E+10	5.54E+09	4.99E+09	5.02E+09	4.75E+09	4.74E+09	4.99E+09	4.01E+09	3.91E+09
64	1.32E+10	5.40E+09	5.34E+09	5.42E+09	6.07E+09	1.05E+10	4.53E+09	5.42E+09	4.94E+09	4.24E+09	3.83E+09	5.42E+09	4.81E+09	3.58E+09
128	1.16E+10	5.78E+09	4.53E+09	5.79E+09	5.92E+09	7.64E+09	3.82E+09	5.79E+09	5.34E+09	3.48E+09	3.25E+09	5.79E+09	5.20E+09	2.95E+09

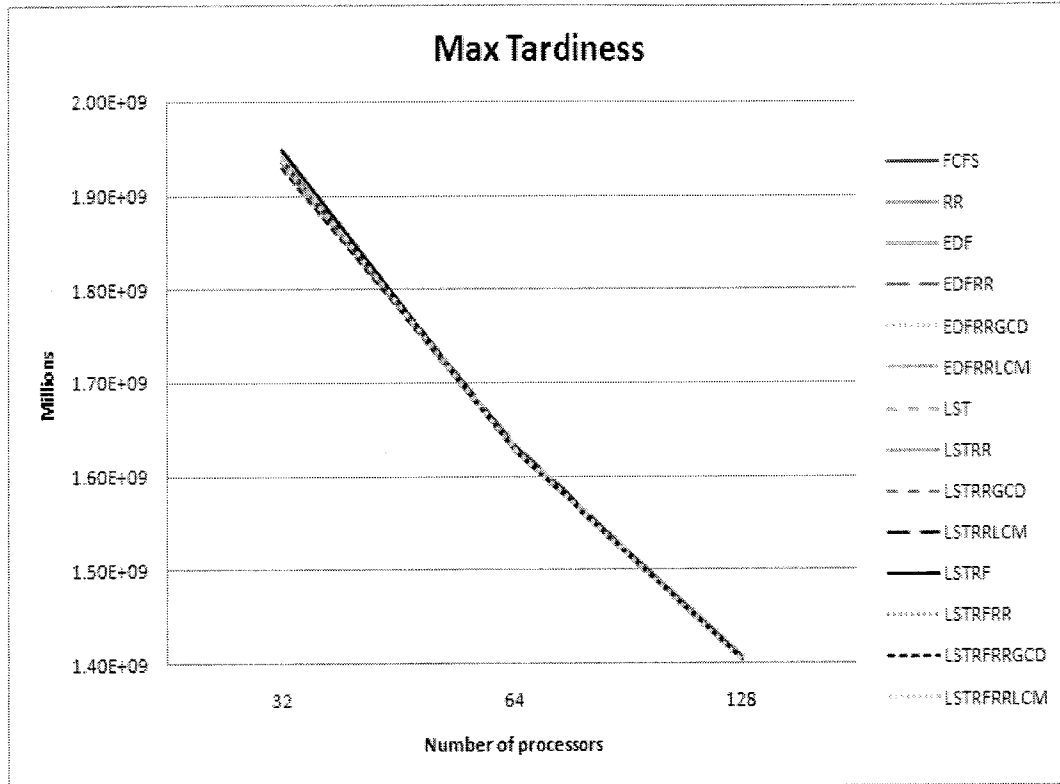


Figure 4.18: Maximum Tardiness

Figure 4.17 show's that maximum tardiness is not fixed it varies on the workload. However, it can be observed that LSTRF, LST, RR, LSTRR, LSTRRGCD, LSTRRLCM, EDFRR, EDFRRGCD, EDFRRLCM, LSTRFRR, LSTRFRRGCD, LSTRFRRLCM and FCFS showed a sharp fall from 32 to 64 and 128 processors as well as overlap one another. However, there is a slight difference between the algorithms. Results showed that LSTRFRRLCM and LSTRF have the best performance, while EDF and EDFRRLCM showed the worst performance.

To ensure that the value obtained is consistent, standard deviation of each algorithms turnaround time of each set of experiments, based on 32, 64 and 128

processors were computed. Table 4.18, showed the standard deviation of Figure 4.18 results.

Table 4.18: Standard Deviation

Standard Deviation	FCFS	RR	EDF	EDFRR	EDFRRGCD	EDFRRLCM	LST	LSTRR	LSTRRGCD	LSTRRLCM	LSTRF	LSTRFRR	LSTRFRRGCD	LSTRFRRLCM
32	8.49E+10	8.47E+10	8.52E+10	8.49E+10	8.49E+10	8.49E+10	8.53E+10	8.49E+10	8.47E+10	8.49E+10	8.54E+10	8.49E+10	8.49E+10	8.49E+10
64	1.03E+11	1.03E+11	1.03E+11	1.03E+11	1.03E+11	1.03E+11	1.03E+11	1.03E+11	1.03E+11	1.03E+11	1.03E+11	1.03E+11	1.03E+11	1.03E+11
128	1.26E+11	1.26E+11	1.26E+11	1.26E+11	1.26E+11	1.26E+11	1.26E+11	1.26E+11	1.26E+11	1.26E+11	1.26E+11	1.26E+11	1.26E+11	1.26E+11

4.2.7 Job Variations of Das-2 Traces

As explained in the methodology, scalability test of scheduling algorithms under an increasing real workload were incorporated and as previously explained, ten (10) data sets were formed by using 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 98% of the traces files workload, 100000, 200000, 300000, 400000, 500000, 600000, 700000, 800000, 900000 and 1000000 processes, respectively.

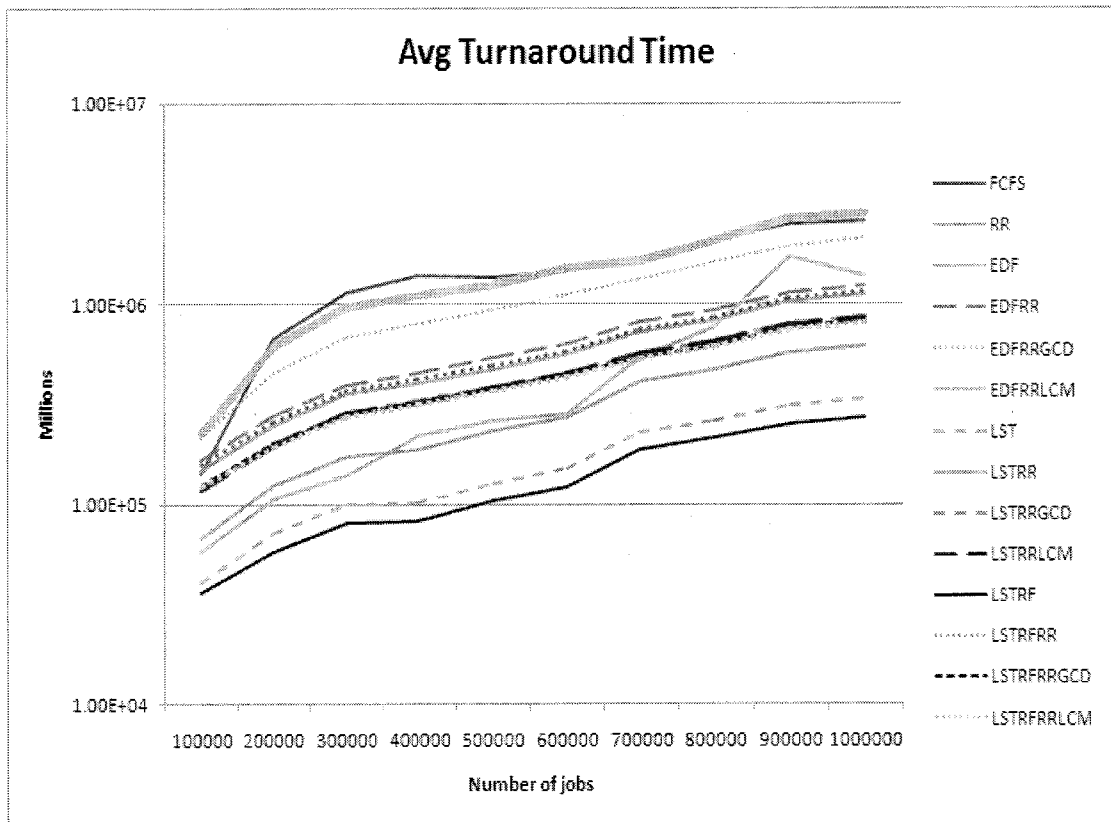


Figure 4.19: Average Turnaround Time

This experiment was carried out using 64 numbers of processors. However, as the number of jobs is increasing, average turnaround time increased. Based on the observation of Figure 4.19, it is clear that LSTRR, LSTRRGCD, LSTRRLCM, EDFRR, EDFRRGCD, LSTRFRR, LSTRFRRGCD, and LSTRFRRLCM showed a smooth and steady rise from 100000 to 1000000 number of jobs as well as overlap one another. However, there is a slight difference between the algorithms. LSTRF, LST, RR, results showed a sharp rise from 100000 to 200000 number of jobs, from 200000 to 300000 number of jobs, and a sharp fall from 300000 to 400000 number of jobs, then sharp fall from 400000 to 600000 number of jobs, then a sharp rise from 600000 to 700000 number of jobs and a sharp rise from 700000 to 1000000 number of jobs. The possible explanation that could be adduced for this observed trend may be due the varying number of jobs. EDF has shown a sharp rise from 100000 to 200000 number of jobs, and a slight fall from 200000 to 300000 number of jobs, and a sharp rise from 300000 to 400000 number of jobs, then sharp fall from 400000 to 600000 number of jobs, then a sharp rise from 600000 to 700000 number of jobs and a sharp rise from 700000 to 800000 number of jobs, then a sharp rise from 800000 to 900000 number of jobs, and finally a sharp fall from 900000 to 1000000 number of jobs, possible reason may be due the varying number of jobs. Meanwhile, FCFS and EDFRRLCM showed a sharp rise from 100000 to 200000 number of jobs, from 200000 to 300000 number of jobs, and a sharp fall from 300000 to 400000 number of jobs, then sharp rise from 400000 to 500000 number of jobs, then a sharp rise from 500000 to 900000 number of jobs and a slight fall from 900000 to 1000000 number of jobs. Results showed that LSTRF and LST have the best performance, while EDFRRLCM and FCFS showed the worst performance.

To ensure that the value obtained is consistent, standard deviation of each algorithms turnaround time of each set of experiments, based on 64 procesors executing 100000 to 1000000 variations of jobs were computed. Table 4.19, showed the standard deviation of Figure 4.19 results.

Table 4.19: Standard Deviation

Standard Deviation	FCFS	RR	EDF	EDFRR	EDFRRGCD	EDFRRLCM	LST	LSTRR	LSTRRGCD	LSTRRLCM	LSTRF	LSTRFR	LSTRFRGCD	LSTRFRRLCM
100000	8.96E+06	4.24E+06	3.63E+06	1.05E+07	1.34E+07	1.42E+07	2.53E+06	9.24E+06	7.44E+06	7.41E+06	2.27E+06	1.00E+07	7.76E+06	7.77E+06
200000	4.18E+07	7.86E+06	6.67E+06	1.75E+07	2.82E+07	3.84E+07	4.49E+06	1.66E+07	1.28E+07	1.23E+07	3.61E+06	1.64E+07	1.27E+07	1.22E+07
300000	7.13E+07	1.08E+07	8.79E+06	2.50E+07	4.28E+07	6.05E+07	6.30E+06	2.24E+07	1.82E+07	1.78E+07	5.08E+06	2.34E+07	1.79E+07	1.73E+07
400000	8.66E+07	1.17E+07	1.37E+07	2.85E+07	5.02E+07	6.94E+07	6.46E+06	2.53E+07	2.07E+07	2.04E+07	5.22E+06	2.65E+07	2.04E+07	1.99E+07
500000	8.60E+07	1.46E+07	1.63E+07	3.38E+07	5.94E+07	7.87E+07	8.00E+06	2.99E+07	2.43E+07	2.40E+07	6.55E+06	3.13E+07	2.38E+07	2.34E+07
600000	9.24E+07	1.70E+07	1.79E+07	3.99E+07	7.02E+07	9.44E+07	9.54E+06	3.52E+07	2.84E+07	2.82E+07	7.70E+06	3.67E+07	2.77E+07	2.73E+07
700000	1.05E+08	2.61E+07	3.41E+07	5.11E+07	8.43E+07	1.04E+08	1.44E+07	4.58E+07	3.57E+07	3.55E+07	1.18E+07	5.31E+07	3.41E+07	3.38E+07
800000	1.30E+08	2.99E+07	4.81E+07	5.88E+07	1.02E+08	1.30E+08	1.69E+07	5.26E+07	4.10E+07	4.07E+07	1.34E+07	5.43E+07	3.90E+07	3.86E+07
900000	1.57E+08	3.58E+07	1.08E+08	7.14E+07	1.23E+08	1.67E+08	1.97E+07	6.44E+07	5.04E+07	5.00E+07	1.59E+07	6.64E+07	4.79E+07	4.72E+07
1000000	1.64E+08	3.86E+07	8.64E+07	7.77E+07	1.35E+08	1.79E+08	2.12E+07	6.99E+07	5.45E+07	5.42E+07	1.73E+07	7.19E+07	5.18E+07	5.11E+07

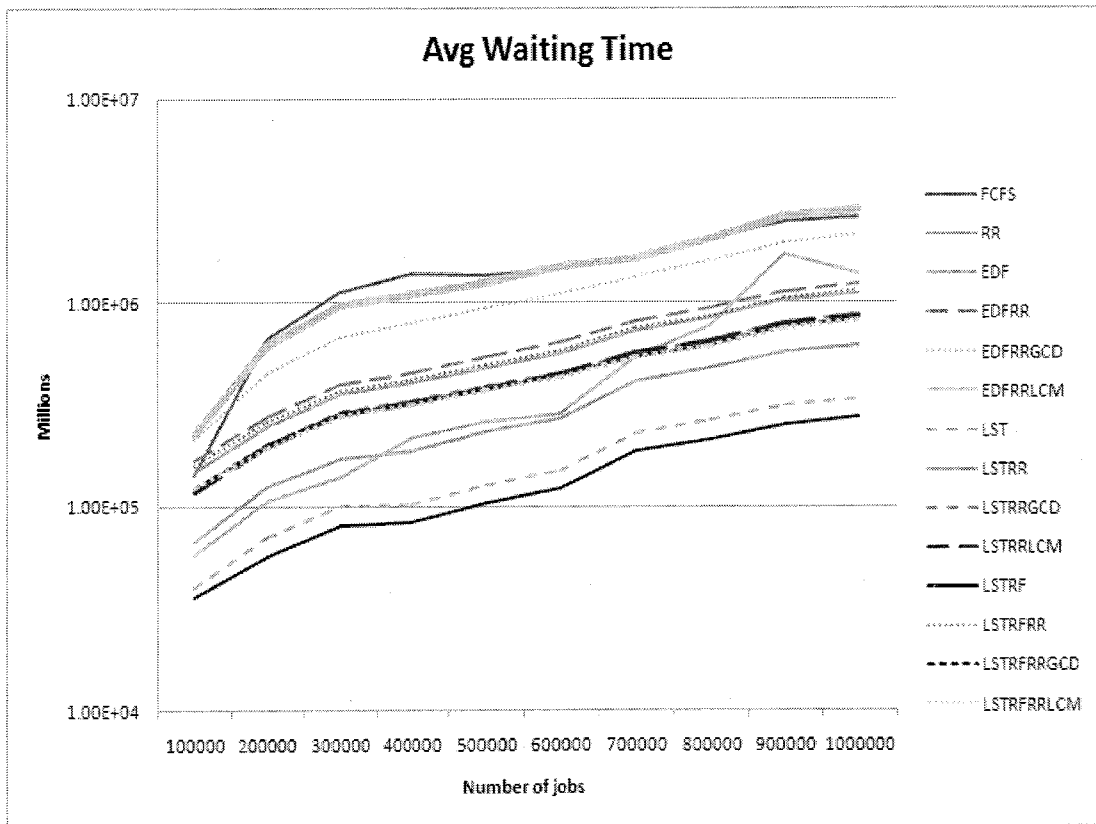


Figure 4.20: Average Waiting Time

It can be observed from Figure 4.20 that LSTRR, LSTRRGCD, LSTRRLCM, EDFRR, EDFRRGCD, LSTRFR, LSTRFRGCD and LSTRFRRLCM showed a smooth and steady rise from 100000 to 10000000 number of jobs as well as overlap one another. However, there is a slight difference between the algorithms. LSTRF,

LST, RR, results showed a sharp rise from 100000 to 200000 number of jobs, from 200000 to 300000 number of jobs, and a sharp fall from 300000 to 400000 number of jobs, then sharp fall from 400000 to 600000 number of jobs, then a sharp rise from 600000 to 700000 number of jobs and a sharp rise from 700000 to 1000000 number of jobs, possible reason may be due to the varying number of jobs. EDF has shown a sharp rise from 100000 to 200000 number of jobs, and a slight fall from 200000 to 300000 number of jobs, and a sharp rise from 300000 to 400000 number of jobs, then sharp fall from 400000 to 600000 number of jobs, then a sharp rise from 600000 to 700000 number of jobs and a sharp rise from 700000 to 800000 number of jobs, then a sharp rise from 800000 to 900000 number of jobs, and finally a sharp fall from 900000 to 1000000 number of jobs. Meanwhile, FCFS and EDFRRLCM showed a sharp rise from 100000 to 200000 number of jobs, from 200000 to 300000 number of jobs, and a sharp fall from 300000 to 400000 number of jobs, then sharp rise from 400000 to 500000 number of jobs, then a sharp rise from 500000 to 900000 number of jobs and a slight fall from 900000 to 1000000 number of jobs. Results showed that LSTRF and LST have the best performance, while EDFRRLCM and FCFS showed the worst performance.

To ensure that the value obtained is consistent, standard deviation of each algorithms turnaround time of each set of experiments, based on 64 processors executing 100000 to 1000000 variations of jobs were computed. Table 4.20, showed the standard deviation of Figure 4.20 results.

Table 4.20: Standard Deviation

Standard Deviation	FCFS	RR	EDF	EDFR	EDFRFGCD	EDFRRLCM	LST	LSTR	LSTRFGCD	LSTRRLCM	LSTRF	LSTRFR	LSTRFRFGCD	LSTRFRRLCM
100000	8.94E+06	4.23E+06	3.61E+06	1.05E+07	1.34E+07	1.42E+07	2.52E+06	9.23E+06	7.42E+06	7.40E+06	2.26E+06	1.00E+07	7.75E+06	7.76E+06
200000	4.18E+07	7.83E+06	6.64E+06	1.74E+07	2.82E+07	3.83E+07	4.48E+06	1.56E+07	1.27E+07	1.23E+07	3.58E+06	1.64E+07	1.27E+07	1.22E+07
300000	7.12E+07	1.08E+07	8.75E+06	2.50E+07	4.28E+07	6.05E+07	6.26E+06	2.24E+07	1.82E+07	1.78E+07	5.04E+06	2.34E+07	1.78E+07	1.73E+07
400000	8.66E+07	1.17E+07	1.37E+07	2.85E+07	5.02E+07	6.94E+07	6.43E+06	2.53E+07	2.07E+07	2.03E+07	5.19E+06	2.64E+07	2.04E+07	1.98E+07
500000	8.60E+07	1.46E+07	1.65E+07	3.38E+07	5.94E+07	7.87E+07	7.97E+06	2.99E+07	2.43E+07	2.40E+07	6.53E+06	3.12E+07	2.38E+07	2.34E+07
600000	9.24E+07	1.70E+07	1.79E+07	3.99E+07	7.02E+07	9.44E+07	9.51E+06	3.52E+07	2.84E+07	2.82E+07	7.68E+06	3.67E+07	2.77E+07	2.73E+07
700000	1.05E+08	2.61E+07	3.41E+07	5.10E+07	8.43E+07	1.04E+08	1.44E+07	4.58E+07	3.57E+07	3.55E+07	1.18E+07	4.73E+07	3.41E+07	3.38E+07
800000	1.30E+08	2.98E+07	4.80E+07	5.88E+07	1.02E+08	1.30E+08	1.98E+07	5.28E+07	4.10E+07	4.07E+07	1.34E+07	5.42E+07	3.90E+07	3.86E+07
900000	1.57E+08	3.57E+07	1.00E+08	7.14E+07	1.23E+08	1.67E+08	1.97E+07	6.44E+07	5.04E+07	5.00E+07	1.59E+07	6.64E+07	4.79E+07	4.72E+07
1000000	1.54E+08	3.89E+07	8.64E+07	7.77E+07	1.35E+08	1.79E+08	2.12E+07	5.98E+07	5.45E+07	5.42E+07	1.73E+07	7.19E+07	5.18E+07	5.11E+07

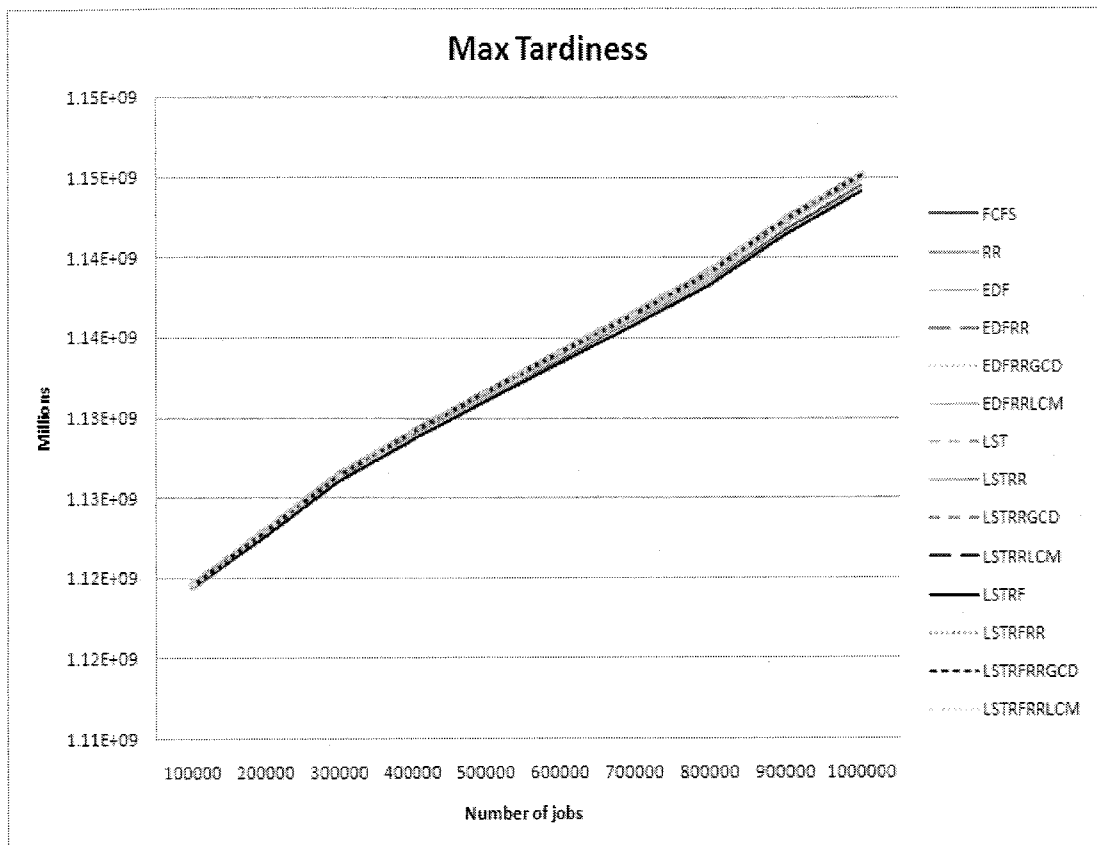


Figure 4.21: Maximum Tardiness

Figure 4.21 show's that maximum tardiness is not fixed it varies on the workload. However, it can be observed from the graph point of view that LSTRF, LST, RR, LSTRR, LSTRRGCD, LSTRRLCM, EDFRR, EDFRRGCD, EDFRRLCM, LSTRFRR, LSTRFRRGCD, LSTRFRRLCM and FCFS showed sharp rise from 100000 to 1000000 number of jobs as well as overlap one another. However, there is a slight difference between the algorithms. Results showed that LSTRF and LST have the best performance, while FCFS and EDFRRLCM showed the worst performance.

To ensure that the value obtained is consistent, standard deviation of each algorithms turnaround time of each set of experiments, based on 64 procesors executing 100000 to 1000000 variations of jobs were computed. Table 4.22, showed the standard deviation of Figure 4.21 results.

Table 4.21: Standard Deviation

Standard Deviation	FCFS	RR	EDF	EDFRR	EDFRRGCD	EDFRRLCM	LST	LSTRR	LSTRRGCD	LSTRRLCM	LSTRF	LSTRFRR	LSTRFRRGCD	LSTRFRRLCM
100000	7.05E+10	7.05E+10	7.05E+10	7.05E+10	7.05E+10	7.05E+10	7.05E+10	7.05E+10	7.05E+10	7.05E+10	7.05E+10	7.05E+10	7.05E+10	7.05E+10
200000	7.07E+10	7.07E+10	7.07E+10	7.07E+10	7.07E+10	7.07E+10	7.07E+10	7.07E+10	7.07E+10	7.07E+10	7.07E+10	7.07E+10	7.07E+10	7.07E+10
300000	7.09E+10	7.09E+10	7.09E+10	7.10E+10	7.10E+10	7.10E+10	7.09E+10	7.10E+10	7.10E+10	7.10E+10	7.09E+10	7.10E+10	7.10E+10	7.10E+10
400000	7.11E+10	7.11E+10	7.11E+10	7.11E+10	7.11E+10	7.11E+10	7.11E+10	7.11E+10	7.11E+10	7.11E+10	7.11E+10	7.11E+10	7.11E+10	7.11E+10
500000	7.13E+10	7.13E+10	7.12E+10	7.13E+10	7.13E+10	7.13E+10	7.12E+10	7.13E+10	7.13E+10	7.13E+10	7.12E+10	7.13E+10	7.13E+10	7.13E+10
600000	7.14E+10	7.14E+10	7.14E+10	7.14E+10	7.14E+10	7.14E+10	7.14E+10	7.14E+10	7.14E+10	7.14E+10	7.14E+10	7.14E+10	7.14E+10	7.14E+10
700000	7.16E+10	7.16E+10	7.16E+10	7.16E+10	7.16E+10	7.16E+10	7.15E+10	7.16E+10	7.16E+10	7.16E+10	7.16E+10	7.16E+10	7.16E+10	7.16E+10
800000	7.17E+10	7.17E+10	7.17E+10	7.17E+10	7.17E+10	7.17E+10	7.17E+10	7.17E+10	7.17E+10	7.17E+10	7.17E+10	7.17E+10	7.17E+10	7.17E+10
900000	7.19E+10	7.19E+10	7.20E+10	7.20E+10	7.20E+10	7.20E+10	7.19E+10	7.20E+10	7.20E+10	7.20E+10	7.19E+10	7.20E+10	7.20E+10	7.20E+10
1000000	7.21E+10	7.21E+10	7.21E+10	7.21E+10	7.21E+10	7.21E+10	7.21E+10	7.21E+10	7.21E+10	7.21E+10	7.21E+10	7.21E+10	7.21E+10	7.21E+10

In contrast to Figure 4.21, experiment was carried out by varying the number from 64 processors to 128 numbers of processors.

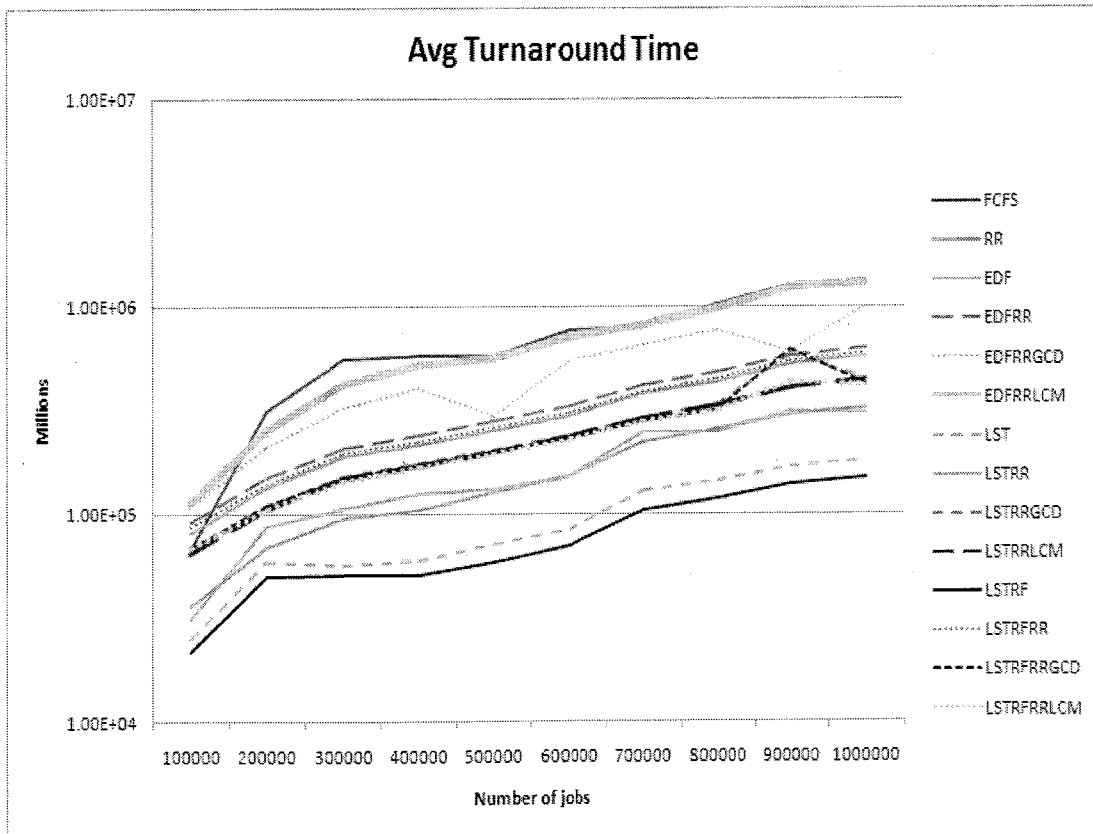


Figure 4.22: Average Turnaround Time

However, it can be observed that Figure 4.22 showed that LSTRR, LSTRRLCM, EDFRR, EDFRRGCD, LSTRFRR, and LSTRFRRLCM are smooth and steady rise from 100000 to 1000000 number of jobs as well as overlap one another. However, there is a slight difference between the algorithms. LSTRF, LST, RR, results, showed a sharp rise from 100000 to 200000 number of jobs, and a sharp fall from 200000 to 400000 number of jobs, and a sharp rise from 400000 to 600000 number of jobs, then sharp rise from 400000 to 600000 number of jobs, from 600000 to 700000 number of jobs, then from 700000 to 900000 number of jobs, then from 900000 to 1000000 number of jobs. EDF has shown a sharp rise from 100000 to 200000 number of jobs, and a slight rise from 200000 to 400000 number of jobs, and a slight fall from 400000 to 600000 number of jobs, then sharp rise from 600000 to 700000 number of jobs, then sharp fall from 700000 to 800000 number of jobs, then a slight rise from 800000 to 900000 number of jobs and finally a slight fall from 900000 to 1000000 number of jobs, possible reason may be due the varying number of jobs. LSTRFRRGCD and LSTRRGCD showed a smooth and steady rise from 100000 to 800000 number of jobs, then a sharp rise from 800000 to 900000 number of jobs, probable reason may be due to heavy workload, then a sharp fall from 900000 to 1000000 number of jobs. EDFRRGCD showed a smooth and steady rise from 100000 to 400000 number of jobs, then a sharp fall from 400000 to 500000 number of jobs, then a sharp rise from 500000 to 600000 number of jobs, then smooth and steady from 600000 to 800000 number of jobs, then a sharp fall from 800000 to 900000 number of jobs, then finally a sharp rise from 900000 to 1000000 number of jobs. Meanwhile, FCFS and EDFRRLCM showed a sharp rise from 100000 to 200000 number of jobs, from 200000 to 300000 number of jobs, and a sharp fall from 300000 to 400000 number of jobs, then sharp rise from 400000 to 500000 number of jobs, then a sharp rise from 500000 to 900000 number of jobs and a slight fall from 900000 to 1000000 number of jobs. Results showed that LSTRF and LST have the best performance, while FCFS and EDFRRLCM showed the worst performance.

To ensure that the value obtained is consistent, standard deviation of each algorithms turnaround time of each set of experiments, based on 128 procesors executing 100000 to 1000000 variations of jobs were computed. Table 4.22, showed the standard deviation of Figure 4.22 results.

Table 4.22: Standard Deviation

Standard Deviation	FCFS	RR	EDF	EDFRR	EDFRRGCD	EDFRRLCM	LST	LSTRR	LSTRRGCD	LSTRRLCM	LSTRF	LSTRFRR	LSTRFRRGCD	LSTRFRRLCM
100000	6.12E+06	3.25E+06	2.82E+06	8.19E+06	9.77E+06	1.02E+07	2.27E+06	7.26E+06	5.99E+06	5.82E+06	1.97E+06	7.88E+06	6.20E+06	6.18E+06
200000	2.80E+07	6.21E+06	7.88E+06	1.33E+07	1.88E+07	2.28E+07	5.24E+06	1.20E+07	9.77E+06	9.38E+06	4.52E+06	1.26E+07	9.84E+06	9.37E+06
300000	4.99E+07	8.56E+06	9.47E+06	1.88E+07	2.92E+07	3.77E+07	5.08E+06	1.69E+07	1.36E+07	1.32E+07	4.58E+06	1.79E+07	1.34E+07	1.28E+07
400000	5.14E+07	9.44E+06	1.12E+07	2.15E+07	3.61E+07	4.70E+07	5.34E+06	1.93E+07	1.57E+07	1.54E+07	4.53E+06	2.01E+07	1.56E+07	1.50E+07
500000	5.14E+07	1.15E+07	1.19E+07	2.53E+07	4.11E+07	5.07E+07	6.38E+06	2.28E+07	1.82E+07	1.79E+07	5.30E+06	2.35E+07	1.78E+07	1.73E+07
600000	6.87E+07	1.36E+07	1.37E+07	2.98E+07	4.88E+07	6.29E+07	7.98E+06	2.66E+07	2.13E+07	2.10E+07	6.28E+06	2.78E+07	2.08E+07	2.04E+07
700000	7.09E+07	1.89E+07	2.20E+07	3.77E+07	5.88E+07	7.27E+07	1.16E+07	3.40E+07	2.62E+07	2.60E+07	9.32E+06	3.58E+07	2.51E+07	2.47E+07
800000	9.28E+07	2.30E+07	2.20E+07	4.31E+07	6.94E+07	8.77E+07	1.30E+07	3.88E+07	3.00E+07	2.96E+07	1.06E+07	4.00E+07	2.86E+07	2.81E+07
900000	1.14E+08	2.72E+07	2.81E+07	5.20E+07	5.48E+07	1.11E+08	1.53E+07	4.72E+07	5.28E+07	3.63E+07	1.24E+07	4.86E+07	5.51E+07	3.92E+07
1000000	1.20E+08	2.92E+07	2.77E+07	5.68E+07	9.07E+07	1.18E+08	1.65E+07	5.17E+07	4.03E+07	3.99E+07	1.35E+07	5.31E+07	3.84E+07	3.77E+07

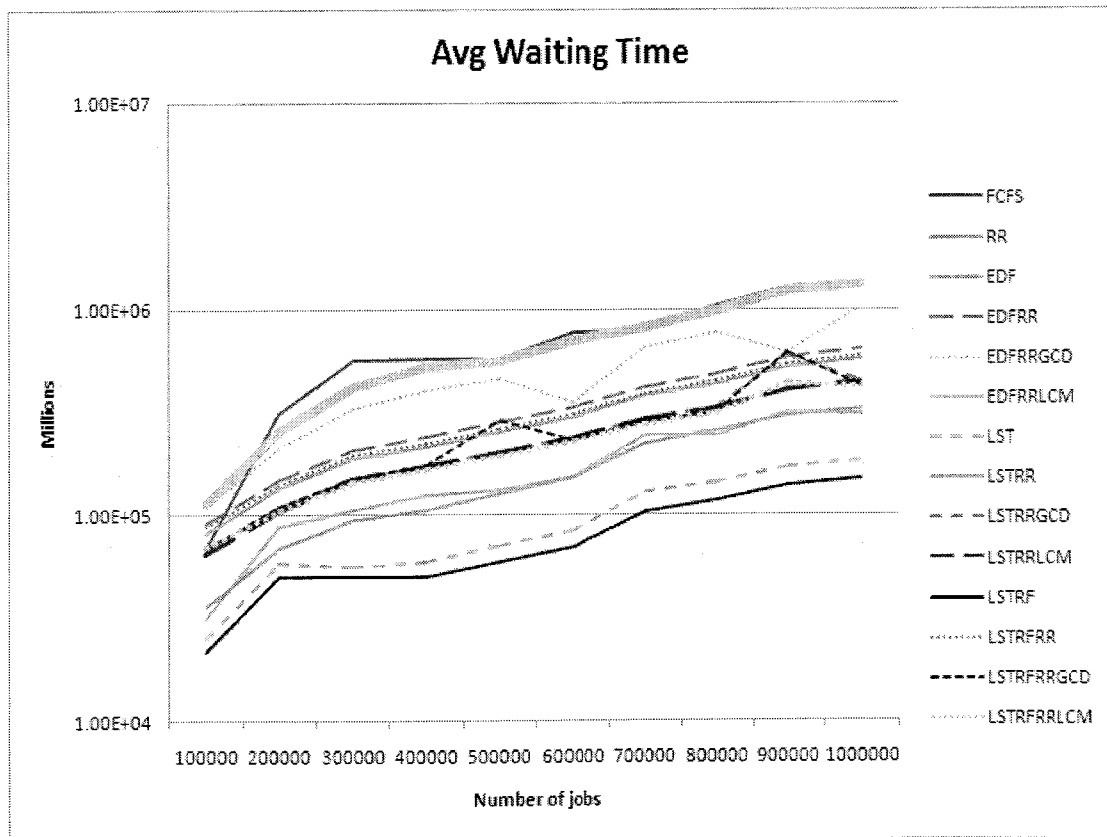


Figure 4.23: Average Waiting Time

It can be observed from Figure 4.23, that LSTRR, LSTRRLCM, EDFRR,

EDFRRGCD, LSTRFRR, and LSTRFRRLCM are smooth and steady rise from 100000 to 1000000 number of jobs as well as overlap one another. However, there is a slight difference between the algorithms. LSTRF, LST, RR, results, showed a sharp rise from 100000 to 200000 number of jobs, and a sharp fall from 200000 to 400000 number of jobs, and a sharp rise from 400000 to 600000 number of jobs, then sharp rise from 400000 to 600000 number of jobs, from 600000 to 700000 number of jobs, then from 700000 to 900000 number of jobs, then from 900000 to 1000000 number of jobs, possible reason may be due the varying number of jobs. EDF has shown a sharp rise from 100000 to 200000 number of jobs, and a slight rise from 200000 to 400000 number of jobs, and a slight fall from 400000 to 600000 number of jobs, then sharp rise from 600000 to 700000 number of jobs, then sharp fall from 700000 to 800000 number of jobs, then a slight rise from 800000 to 900000 number of jobs and finally a slight fall from 900000 to 1000000 number of jobs. LSTRFRRGCD and LSTRRGCD showed a smooth and steady rise from 100000 to 800000 number of jobs, then a sharp rise from 800000 to 900000 number of jobs, probable reason may be due to heavy workload, then a sharp fall from 900000 to 1000000 number of jobs. EDFRRGCD showed a smooth and steady rise from 100000 to 400000 number of jobs, then a sharp fall from 400000 to 500000 number of jobs, then a sharp rise from 500000 to 600000 number of jobs, then smooth and steady from 600000 to 800000 number of jobs, then a sharp fall from 800000 to 900000 number of jobs, then finally a sharp rise from 900000 to 1000000 number of jobs. Meanwhile, FCFS and EDFRRLCM showed a sharp rise from 100000 to 200000 number of jobs, from 200000 to 300000 number of jobs, and a sharp fall from 300000 to 400000 number of jobs, then sharp rise from 400000 to 500000 number of jobs, then a sharp rise from 500000 to 900000 number of jobs and a slight fall from 900000 to 1000000 number of jobs. Results showed that LSTRF and LST have the best performance, while FCFS and EDFRRLCM showed the worst performance.

To ensure that the value obtained is consistent, standard deviation of each algorithms turnaround time of each set of experiments, based on 128 procesors executing 100000 to 1000000 variations of jobs were computed. Table 4.23, showed the standard deviation of Figure 4.23 results.

Table 4.23: Standard Deviation

Standard Deviation	FCFS	RR	EDF	EDFRR	EDFRRGCD	EDFRRLCM	LST	LSTRR	LSTRRGCD	LSTRRLCM	LSTRF	LSTRFRR	LSTRFRRGCD	LSTRFRRLCM
100000	6.10E+06	3.23E+06	2.79E+06	8.17E+06	9.75E+06	1.02E+07	2.26E+06	7.24E+06	5.97E+06	5.80E+06	1.95E+06	7.06E+06	6.18E+06	6.15E+06
200000	2.79E+07	6.16E+06	7.83E+06	1.33E+07	1.87E+07	2.27E+07	5.20E+06	1.19E+07	9.72E+06	9.34E+06	4.47E+06	1.25E+07	9.79E+06	9.33E+06
300000	4.99E+07	8.51E+06	9.43E+06	1.86E+07	2.92E+07	3.76E+07	5.03E+06	1.69E+07	1.35E+07	1.32E+07	4.51E+06	1.74E+07	1.33E+07	1.28E+07
400000	5.14E+07	9.41E+06	1.12E+07	2.15E+07	3.60E+07	4.69E+07	5.30E+06	1.93E+07	1.57E+07	1.53E+07	4.49E+06	2.00E+07	1.55E+07	1.50E+07
500000	5.13E+07	1.15E+07	1.18E+07	2.53E+07	4.10E+07	5.06E+07	6.36E+06	2.28E+07	1.81E+07	1.78E+07	5.26E+06	2.35E+07	1.77E+07	1.73E+07
600000	6.87E+07	1.36E+07	1.37E+07	2.98E+07	3.14E+07	6.29E+07	7.52E+06	2.68E+07	2.13E+07	2.10E+07	6.22E+06	2.76E+07	2.07E+07	2.04E+07
700000	7.08E+07	1.99E+07	2.20E+07	3.76E+07	5.87E+07	7.27E+07	1.15E+07	3.39E+07	2.82E+07	2.60E+07	9.29E+06	3.50E+07	2.51E+07	2.47E+07
800000	9.25E+07	2.29E+07	2.20E+07	4.30E+07	6.93E+07	8.77E+07	1.30E+07	3.87E+07	3.00E+07	2.96E+07	1.06E+07	3.99E+07	2.85E+07	2.80E+07
900000	1.14E+08	2.72E+07	2.80E+07	5.20E+07	5.48E+07	1.11E+08	1.52E+07	4.71E+07	5.28E+07	3.62E+07	1.24E+07	4.85E+07	5.51E+07	3.91E+07
1000000	1.20E+08	2.92E+07	2.77E+07	5.68E+07	9.07E+07	1.18E+08	1.84E+07	5.18E+07	4.03E+07	3.99E+07	1.34E+07	5.30E+07	3.83E+07	3.77E+07

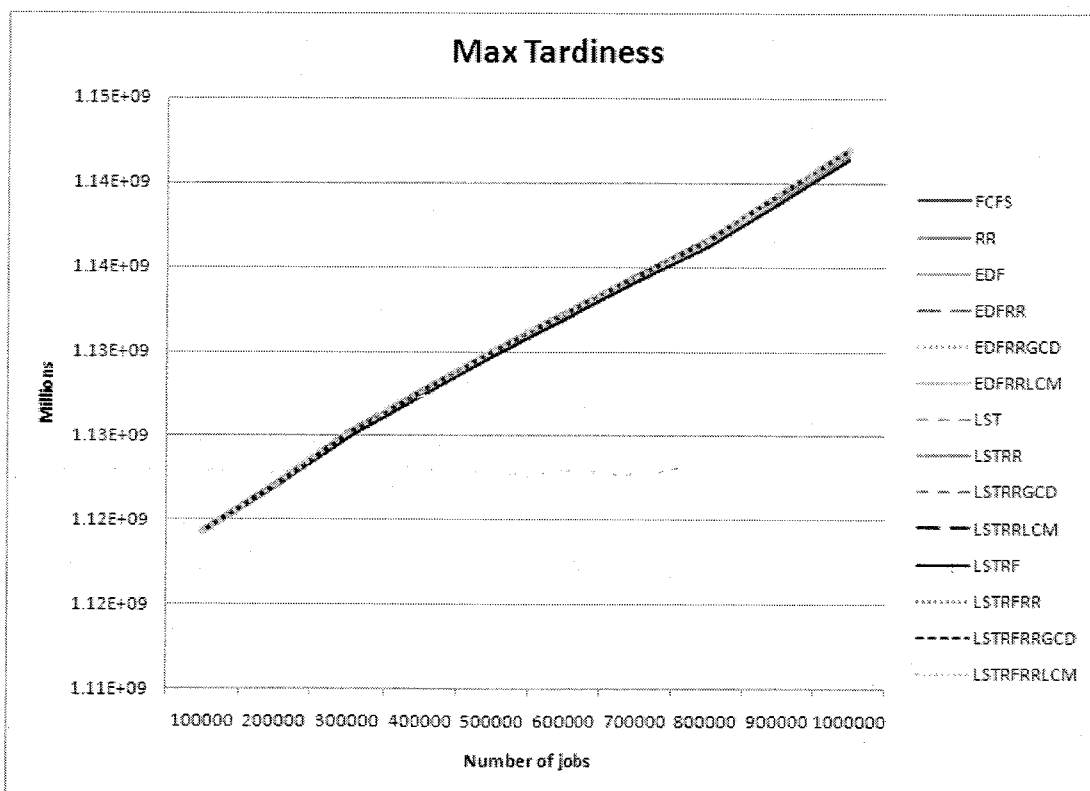


Figure 4.24: Maximum Tardiness

Figure 4.24 shows that maximum tardiness is not fixed it vary on the workload. However, it can be observed from the graph point of view that LSTRF, LST, RR, LSTRR, LSTRRGCD, LSTRRLCM, EDFRR, EDFRRGCD, EDFRRLCM,

LSTRFRR, LSTRFRRGCD, LSTRFRRLCM and FCFS showed a sharp rise from 100000 to 1000000 number of jobs as well as overlap one another. However, there is a slight difference between the algorithms. Results showed that LSTRF and LST have the best performance, while LSTRFLCM and EDFRRLCM showed the worst performance.

To ensure that the value obtained is consistent, standard deviation of each algorithms turnaround time of each set of experiments, based on 128 processors executing 100000 to 1000000 variations of jobs were computed. Table 4.24, showed the standard deviation of Figure 4.24 results.

Table 4.24: Standard Deviation

Standard Deviation	FCFS	RR	EDF	EDFRR	EDFRRGCD	EDFRRLCM	LST	LSTR	LSTRGCD	LSTRRLCM	LSTRF	LSTRFRR	LSTRFRRGCD	LSTRFRRLCM
100000	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11
200000	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11
300000	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11
400000	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11
500000	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11	1.01E+11
600000	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11
700000	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.34E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11
800000	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11
900000	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11
1000000	1.03E+11	1.02E+11	1.02E+11	1.03E+11	1.03E+11	1.03E+11	1.02E+11	1.03E+11	1.03E+11	1.03E+11	1.02E+11	1.03E+11	1.03E+11	1.03E+11

4.2.8 Job Variations Sharcnet Traces

We incorporate scalability test of scheduling algorithms under an increasing real workload. Meanwhile, we formed ten (10) data sets by using 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 98% of the traces files workload, 100000, 200000, 300000, 400000, 500000, 600000, 700000, 800000, 900000 and 1000000 processes, respectively.

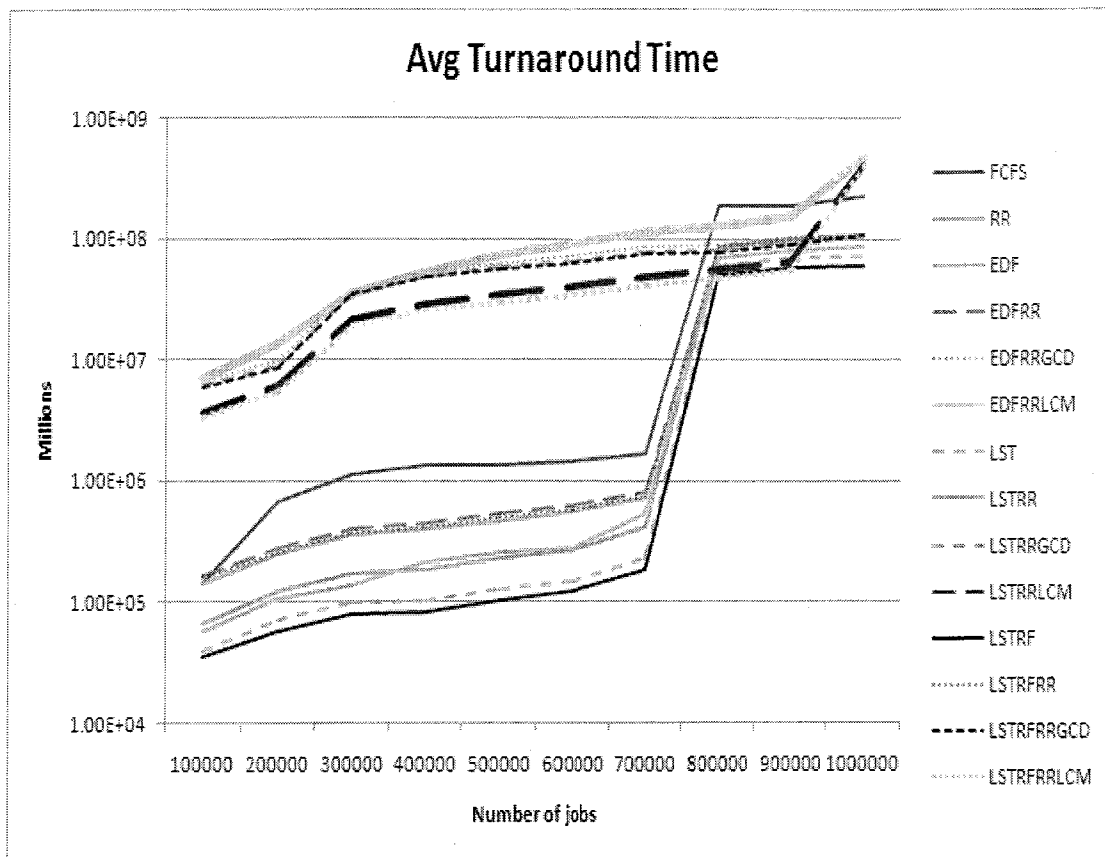


Figure 4.25: Average Turnaround Time

This experiment was carried out using 64 numbers of processors. However, as the number of jobs increased, average turnaround time increased as well. Based on the general observation of Figure 4.25, LSTRRGCD, EDFRRGCD, and LSTRFRRGCD showed a sharp rise from 100000 to 200000 number of jobs, then from 200000 to 300000, then a smooth from 300000 to 1000000 as well as overlap one another. However, there is a slight difference between the algorithms. LSTRFRRLCM and LSTRRLCM showed a sharp rise from 100000 to 2000000 jobs, then from 200000 to 300000, then a smooth from 300000 to 900000 number of jobs, then a sharp rise from 900000 to 1000000 number of jobs. LSTRR, EDFRR, and LSTRFRR are smooth and steady from 100000 to 7000000 number of jobs, then a sharp rise from 700000 to 800000 number of jobs, then a slight fall from 800000 to 900000 number of jobs, the finally a sharp rise from 900000 to 1000000 number of jobs. LSTRF, LST, RR, results, showed a sharp rise from 100000 to 200000 number of jobs, from 200000 to 300000 number of jobs, and a sharp fall from 300000 to 400000 number of jobs, then sharp fall from 400000 to 600000 number of jobs, then a sharp rise from 600000 to

700000 number of jobs and a sharp rise from 700000 to 800000 number of jobs, then a slight fall from 800000 to 900000 number of jobs, then finally a sharp rise from 900000 to 1000000 number of jobs. EDF has shown a sharp rise from 100000 to 200000 number of jobs, and a slight fall from 200000 to 300000 number of jobs, and a sharp rise from 300000 to 400000 number of jobs, then sharp fall from 400000 to 600000 number of jobs, then a sharp rise from 600000 to 700000 number of jobs and a sharp rise from 700000 to 800000 number of jobs, then a slight fall from 800000 to 900000 number of jobs, and finally a sharp rise from 900000 to 1000000 number of jobs. Meanwhile, FCFS a sharp rise from 100000 to 2000000 number of jobs, then from 200000 to 300000, then a smooth from 300000 to 700000 number of jobs, then a sharp rise from 700000 to 800000 number of jobs, then a slight fall from 800000 to 900000 number of jobs, and finally a slight rise from 900000 to 1000000 number of jobs. Results showed that LSTRF and LST have the best performance. While LSTRFRRGCD then EDFRRLCM showed the worst performance.

To ensure that the value obtained is consistent, standard deviation of each algorithms turnaround time of each set of experiments, based on 64 processors executing 100000 to 1000000 variations of jobs were computed. Table 4.25, showed the standard deviation of Figure 4.25 results.

Table 4.25: Standard Deviation

Standard Deviation	FCFS	RR	EDF	EDFRR	EDFRRGCD	EDFRRLCM	LST	LSTR	LSTRRGCD	LSTRRLCM	LSTRF	LSTRFR	LSTRFRRGCD	LSTRFRRLCM
100000	8.96E+06	4.24E+06	3.63E+06	1.06E+07	4.13E+08	4.46E+08	2.53E+06	9.24E+06	3.83E+08	2.34E+08	2.27E+06	1.10E+07	3.77E+08	2.08E+08
200000	4.18E+07	7.68E+06	6.67E+06	1.75E+07	6.34E+08	8.59E+08	4.49E+06	1.56E+07	5.58E+08	3.89E+08	3.61E+06	1.64E+07	5.46E+08	3.44E+08
300000	7.13E+07	1.08E+07	8.79E+06	2.50E+07	2.38E+09	2.20E+09	6.30E+06	2.24E+07	2.25E+09	1.33E+09	5.08E+06	2.34E+07	2.22E+09	1.20E+09
400000	8.65E+07	1.17E+07	1.37E+07	2.86E+07	3.24E+09	3.35E+09	6.46E+06	2.53E+07	3.03E+09	1.81E+09	5.22E+06	2.65E+07	2.98E+09	1.62E+09
500000	8.80E+07	1.48E+07	1.66E+07	3.38E+07	3.91E+09	4.66E+09	8.00E+06	2.99E+07	3.60E+09	2.18E+09	6.55E+06	3.13E+07	3.63E+09	1.86E+09
600000	9.24E+07	1.70E+07	1.79E+07	3.99E+07	4.51E+09	5.75E+09	9.54E+06	3.62E+07	4.06E+09	2.62E+09	7.70E+06	3.67E+07	3.98E+09	2.14E+09
700000	1.05E+08	2.61E+07	3.41E+07	5.11E+07	5.42E+09	7.21E+09	1.44E+07	4.58E+07	4.37E+09	3.04E+09	1.18E+07	4.73E+07	4.77E+09	2.58E+09
800000	1.18E+10	5.42E+09	4.34E+09	5.44E+09	5.75E+09	8.14E+09	3.73E+09	5.46E+09	5.10E+09	3.46E+09	3.19E+09	5.46E+09	4.98E+09	2.98E+09
900000	1.19E+10	6.20E+09	5.09E+09	6.22E+09	6.63E+09	9.69E+09	4.30E+09	6.21E+09	5.77E+09	4.09E+09	3.66E+09	6.23E+09	5.62E+09	3.48E+09
1000000	1.40E+10	6.70E+09	5.54E+09	6.73E+09	6.73E+09	3.03E+10	4.57E+09	6.72E+09	6.73E+09	2.60E+10	3.85E+09	6.72E+09	6.77E+09	2.55E+10

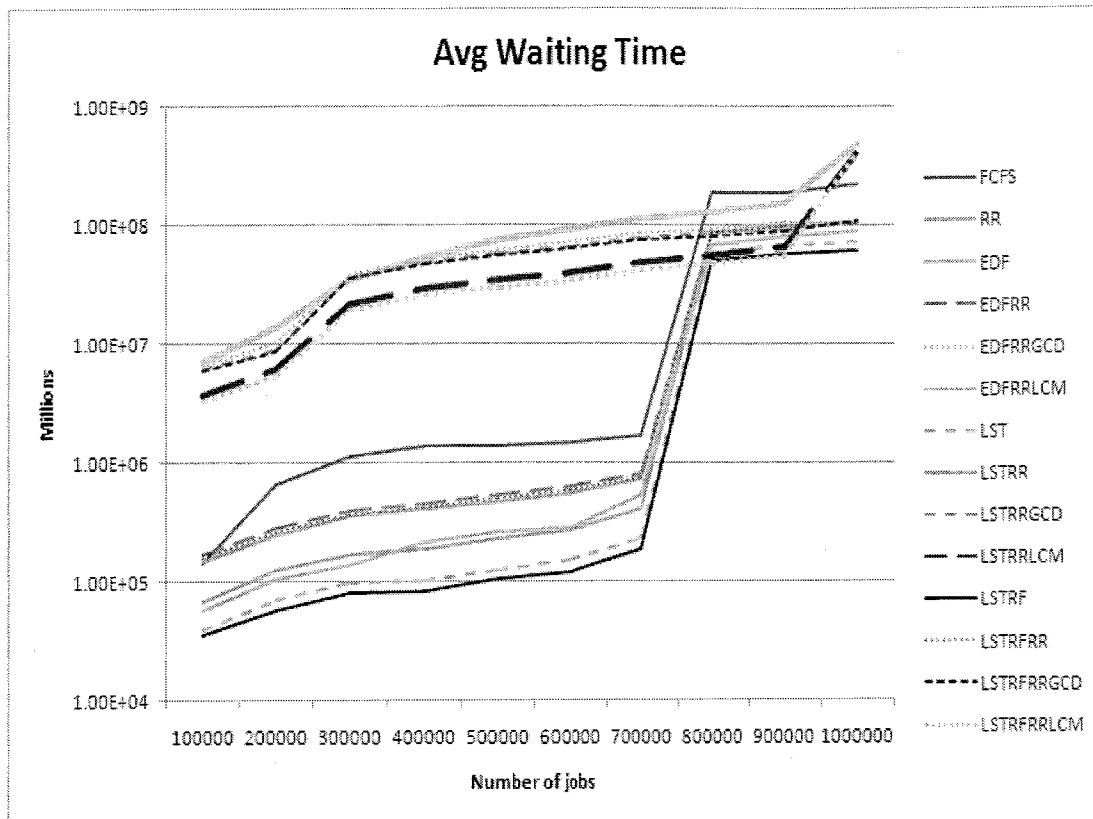


Figure 4.26: Average Waiting Time

It can be observed from Figure 4.26 that LSTRRGCD, EDFRRGCD, and LSTRFRRGCD showed a sharp rise from 100000 to 200000 number of jobs, then from 200000 to 300000, then a smooth from 300000 to 1000000 as well as overlap one another. However, there is a slight difference between the algorithms. LSTRFRRLCM and LSTRRLCM showed a sharp rise from 100000 to 200000 number of jobs, then from 200000 to 300000, then a smooth from 300000 to 900000 number of jobs, then a sharp rise from 900000 to 1000000 number of jobs, probable reason may be due to heavy workload. LSTRR, EDFRR, and LSTRFRR are smooth and steady from 100000 to 700000 number of jobs, then a sharp rise from 700000 to 800000 number of jobs, then a slight fall from 800000 to 900000 number of jobs, the finally a sharp rise from 900000 to 1000000 number of jobs, possible reason may be due to heavy workload. LSTRF, LST, RR, results, showed a sharp rise from 100000 to 200000 number of jobs, from 200000 to 300000 number of jobs, and a sharp fall from 300000 to 400000 number of jobs, then sharp fall from 400000 to 600000 number of jobs, then a sharp rise from 600000 to 700000 number of jobs and a sharp rise from 700000 to 800000 number of jobs, then a slight fall from 800000 to 900000

number of jobs, then finally a sharp rise from 900000 to 1000000 number of jobs, possible reason may be due the varying number of jobs and increase in heavy workload. EDF has shown a sharp rise from 100000 to 200000 number of jobs, and a slight fall from 200000 to 300000 number of jobs, and a sharp rise from 300000 to 400000 number of jobs, then sharp fall from 400000 to 600000 number of jobs, then a sharp rise from 600000 to 700000 number of jobs and a sharp rise from 700000 to 800000 number of jobs, then a slight fall from 800000 to 900000 number of jobs, and finally a sharp rise from 900000 to 1000000 number of jobs. Meanwhile, FCFS a sharp rise from 100000 to 2000000 number of jobs, then from 200000 to 300000, then a smooth from 300000 to 700000 number of jobs, then a sharp rise from 700000 to 800000 number of jobs, then a slight fall from 800000 to 900000 number of jobs, and finally a slight rise from 900000 to 1000000 number of jobs. Results showed that LSTRF and LST have the best performance, while LSTRFRRGCD then EDFRRLCM showed the worst performance.

To ensure that the value obtained is consistent, standard deviation of each algorithms turnaround time of each set of experiments, based on 64 processors executing 100000 to 1000000 variations of jobs were computed. Table 4.26, showed the standard deviation of Figure 4.26 results.

Table 4.26: Standard Deviation

Standard Deviation	FCFS	RR	EDF	EDFRR	EDFRRGCD	EDFRRLCM	LST	LSTRR	LSTRRGCD	LSTRRLCM	LSTRF	LSTRFRR	LSTRFRRGCD	LSTRFRRLCM
100000	8.94E+06	4.23E+06	3.61E+06	1.06E+07	4.12E+06	4.44E+08	2.52E+06	9.23E+06	3.82E+08	2.33E+08	2.26E+06	1.00E+07	3.76E+08	2.07E+08
200000	4.18E+07	7.93E+06	6.64E+06	1.74E+07	6.33E+08	8.58E+08	4.48E+06	1.58E+07	5.58E+08	3.88E+09	3.58E+06	1.64E+07	5.45E+08	3.44E+08
300000	7.12E+07	1.08E+07	8.76E+06	2.50E+07	2.38E+09	2.20E+09	6.26E+06	2.24E+07	2.25E+09	1.33E+09	5.04E+06	2.34E+07	2.21E+09	1.20E+09
400000	8.65E+07	1.17E+07	1.37E+07	2.85E+07	3.24E+09	3.35E+09	6.43E+06	2.53E+07	3.03E+09	1.80E+09	5.19E+06	2.64E+07	2.98E+09	1.62E+09
500000	8.60E+07	1.46E+07	1.66E+07	3.38E+07	3.91E+09	4.65E+09	7.97E+06	2.99E+07	3.58E+09	2.18E+09	6.53E+06	3.12E+07	3.53E+09	1.87E+09
600000	9.24E+07	1.70E+07	1.79E+07	3.99E+07	4.51E+09	5.75E+09	9.51E+06	3.62E+07	4.06E+09	2.51E+09	7.68E+06	3.67E+07	3.98E+09	2.14E+09
700000	1.05E+08	2.61E+07	3.41E+07	5.10E+07	5.42E+09	7.28E+09	1.44E+07	4.58E+07	4.87E+09	3.04E+09	1.18E+07	4.73E+07	4.77E+09	2.58E+09
800000	1.17E+10	5.42E+09	4.34E+09	5.44E+09	5.75E+09	8.14E+09	3.72E+09	5.44E+09	5.44E+09	3.46E+09	3.19E+09	5.45E+09	4.98E+09	2.98E+09
900000	1.19E+10	6.19E+09	5.09E+09	6.21E+09	6.63E+09	9.69E+09	4.30E+09	6.21E+09	5.76E+09	4.08E+09	3.66E+09	6.23E+09	5.62E+09	3.48E+09
1000000	1.40E+10	6.70E+09	5.54E+09	6.73E+09	6.73E+09	3.03E+10	4.57E+09	6.71E+09	6.73E+09	2.60E+10	3.85E+09	6.71E+09	6.77E+09	2.55E+10

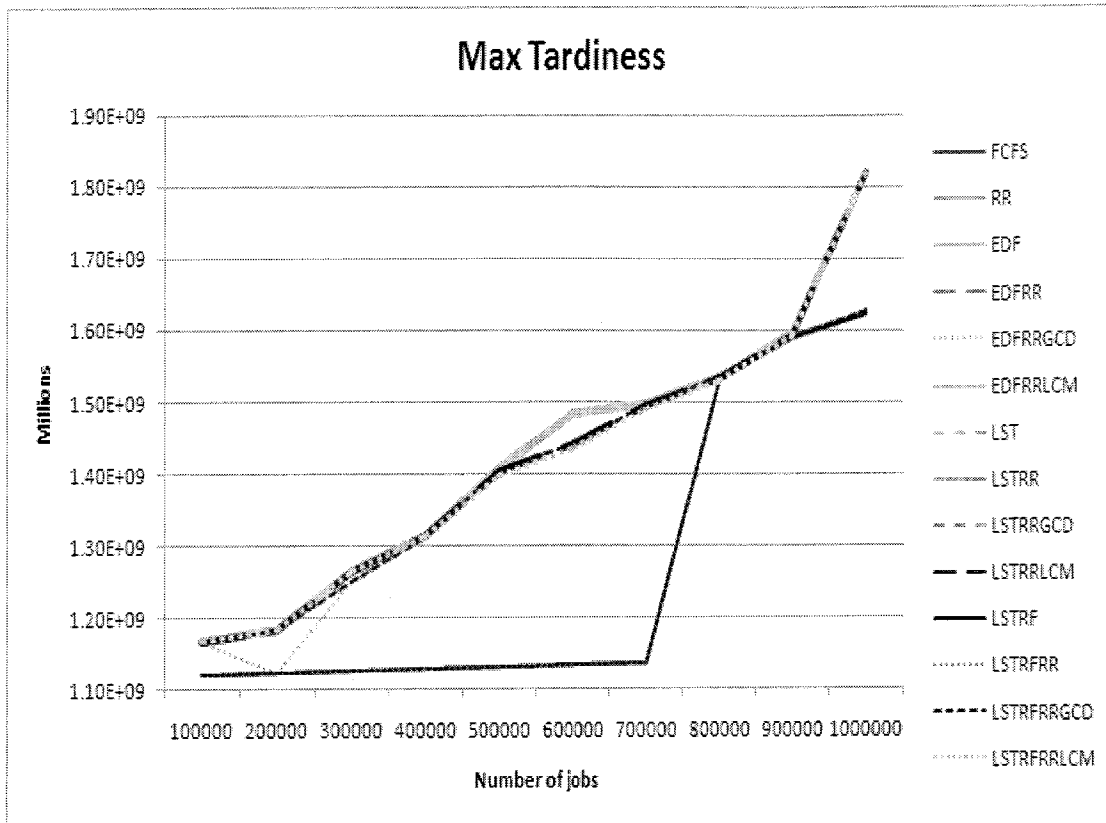


Figure 4.27: Maximum Tardiness

Figure 4.27 shows that maximum tardiness is not fixed it vary on the workload. However, it can be observed that LSTRF, LST, RR, LSTRR, EDFRR, LSTRFRR, FCFS and EDF are overlap one another, showing a sharp and steady rise from 100000 to 700000 numbers of jobs and a high rise from 700000 to 800000 then a high rise from 800000 to 1000000 numbers of jobs. Meanwhile, LSTRRGCD, LSTRFRRGCD, showed sharp rise from 100000 to 200000 number of jobs, from 200000 to 300000 number of jobs, from 300000 to 400000 number of jobs, from 400000 to 500000 number of jobs, then a sharp rise from 500000 to 1000000 number of jobs, possible reason may be due to heavy workload. EDFRRGCD showed sharp fall from 100000 to 200000 number of jobs, possible reason may be due less workload, then a sharp rise from 200000 to 300000 number of jobs, from 300000 to 400000 number of jobs, from 400000 to 500000 number of jobs, then a sharp rise from 500000 to 1000000 number of jobs. LSTRFRRLCM, LSTRRLCM showed sharp rise 100000 to 200000 number of jobs, from 200000 to 300000 number of jobs, from 300000 to 400000 number of jobs, from 400000 to 500000 number of jobs, from 500000 to 1000000 number of jobs, then finally a sharp rise from 900000 to 1000000 number of jobs.

EDFRRLCM showed sharp rise 100000 to 200000 number of jobs, from 200000 to 300000 number of jobs, from 300000 to 400000 number of jobs, from 400000 to 500000 number of jobs, then a sharp rise from 500000 to 600000 number of jobs, a sharp fall from 600000 to 700000 number of jobs, then smooth and steady from 700000 to 1000000 number of jobs. All algorithms seem to be overlapping one another. However, there is a slight difference between the algorithms. Results showed that LSTRF and LST have the best performance, while LSTRFRRGCD and EDFRRLCM showed the worst performance.

To ensure that the value obtained is consistent, standard deviation of each algorithms turnaround time of each set of experiments, based on 64 processors executing 100000 to 1000000 variations of jobs were computed. Table 4.27, showed the standard deviation of Figure 4.27 results.

Table 4.27: Standard Deviation

Standard Deviation	FCFS	RR	EDF	EDFRR	EDFRRGCD	EDFRRLCM	LST	LSTR	LSTRGCD	LSTRRLCM	LSTRF	LSTRFR	LSTRFRGCD	LSTRFRRLCM
100000	7.06E+10	7.06E+10	7.06E+10	7.06E+10	7.36E+10	7.36E+10	7.06E+10	7.06E+10	7.36E+10	7.36E+10	7.06E+10	7.06E+10	7.36E+10	7.36E+10
200000	7.07E+10	7.07E+10	7.07E+10	7.07E+10	7.07E+10	7.46E+10	7.07E+10	7.07E+10	7.46E+10	7.46E+10	7.07E+10	7.07E+10	7.46E+10	7.46E+10
300000	7.09E+10	7.09E+10	7.09E+10	7.10E+10	7.09E+10	7.96E+10	7.09E+10	7.10E+10	7.96E+10	7.96E+10	7.09E+10	7.10E+10	7.96E+10	7.96E+10
400000	7.11E+10	7.11E+10	7.11E+10	7.11E+10	8.27E+10	8.28E+10	7.11E+10	7.11E+10	8.27E+10	8.28E+10	7.11E+10	7.11E+10	8.27E+10	8.28E+10
500000	7.13E+10	7.13E+10	7.12E+10	7.13E+10	8.88E+10	8.87E+10	7.12E+10	7.13E+10	8.88E+10	8.88E+10	7.12E+10	7.13E+10	8.88E+10	8.81E+10
600000	7.14E+10	7.14E+10	7.14E+10	7.14E+10	9.09E+10	9.34E+10	7.14E+10	7.14E+10	9.09E+10	9.09E+10	7.14E+10	7.14E+10	9.09E+10	9.04E+10
700000	7.16E+10	7.16E+10	7.16E+10	7.16E+10	9.41E+10	9.43E+10	7.16E+10	7.16E+10	9.41E+10	9.42E+10	7.16E+10	7.16E+10	9.41E+10	9.39E+10
800000	9.67E+10	9.64E+10	9.66E+10	9.68E+10	9.68E+10	9.68E+10	9.65E+10	9.66E+10	9.66E+10	9.66E+10	9.65E+10	9.66E+10	9.66E+10	9.64E+10
900000	1.00E+11	1.00E+11	1.00E+11	1.00E+11	1.00E+11	1.00E+11	1.00E+11	1.00E+11	1.00E+11	1.00E+11	1.00E+11	1.00E+11	1.00E+11	1.00E+11
1000000	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.16E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.02E+11	1.16E+11

In contrast to Figure 4.25, experiment was carried out by varying the number from 64 processors to 128 numbers of processors.

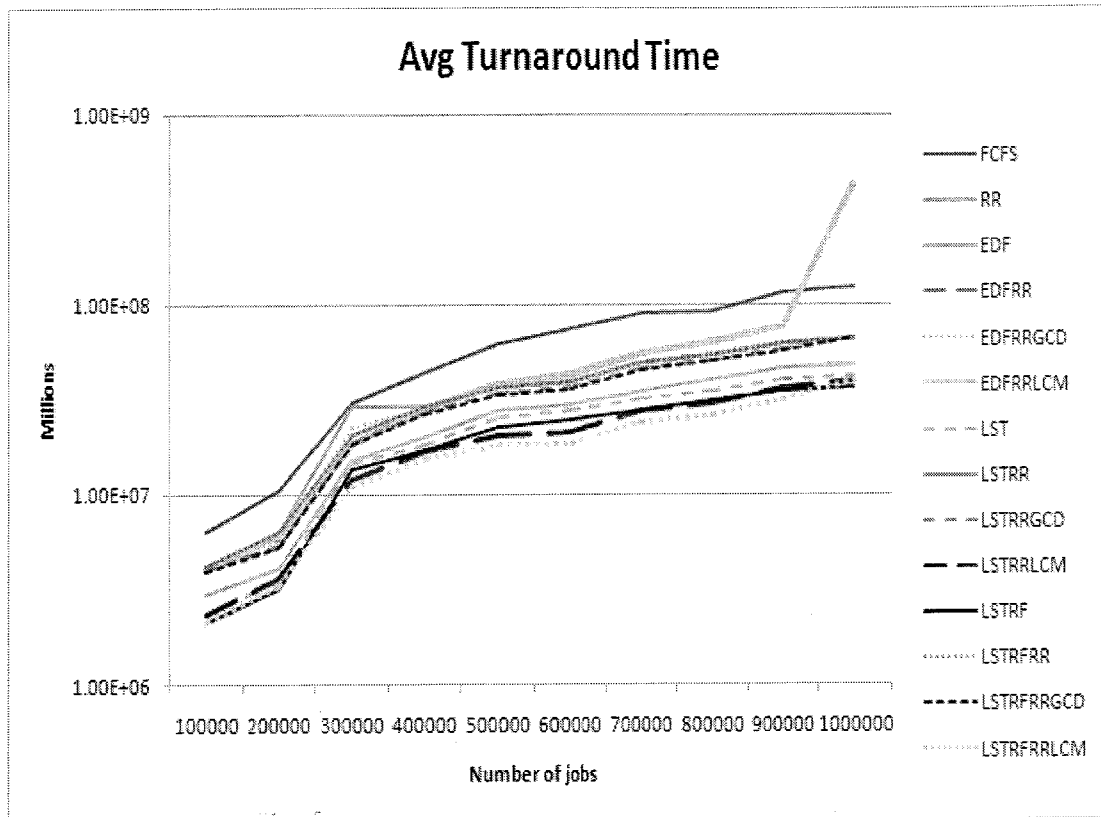


Figure 4.28: Average Turnaround Time

However, it can also be observed that from the graph, as the number of jobs is increasing, average turnaround time is increasing. Based on the general observation of Figure 4.28, LST, RR, LSTRR, LSTRRGCD, EDFRRGCD, LSTRFRR, LSTRFRRGCD, LSTRFRRLCM, and LSTRRLCM showed sharp rise from 100000 to 200000 number of jobs, from 200000 to 300000, then a slight rise from 300000 to 500000 number of jobs, then a slight fall from 500000 to 600000 number of jobs, then a slight rise from 600000 to 700000 number of jobs, finally smooth and steady from 700000 to 1000000 number of jobs. LSTRF showed sharp rise from 100000 to

2000000 number of jobs, from 200000 to 300000, then a slight rise from 300000 to 500000 number of jobs, then finally smooth and steady from 500000 to 1000000 number of jobs. EDFRR showed sharp rise from 100000 to 2000000 number of jobs, from 200000 to 300000, then a slight fall from 300000 to 400000 number of jobs, then a slight rise from 400000 to 500000 number of jobs, then a slight fall from 500000 to 600000 number of jobs, then a slight rise from 600000 to 700000 number of jobs, finally smooth and steady from 700000 to 1000000 number of jobs. EDFRRLCM showed sharp rise from 100000 to 2000000 number of jobs, from 200000 to 300000, then a slight rise from 300000 to 500000 number of jobs, then a slight fall from 500000 to 600000 number of jobs, then a slight rise from 600000 to 700000 number of jobs, then smooth and steady from 700000 to 900000 number of jobs, then finally a sharp rise from 700000 to 1000000 number of jobs, possible reason may be due to heavy workload. Meanwhile, FCFS showed sharp rise from 100000 to 2000000 number of jobs, from 200000 to 300000, then a slight rise from 300000 to 500000 number of jobs, then a slight rise from 500000 to 700000 number of jobs, then a slight fall from 700000 to 800000 number of jobs, then a slight fall from 800000 to 900000 number of jobs, then finally slight rise from 900000 to 1000000 number of jobs.

Moreover, RR, LSTRR, LSTRRGCD, EDFRR, EDFRRGCD, LSTRFRR, LSTRFRRGCD, LSTRFRRLCM and LSTRRLCM are overlapping one another. But there is a slight difference between the algorithms. Results showed that LSTRFRRLCM and LSTRRLCM have the best performance, while FCFS and EDFRRLCM showed the worst performance.

To ensure that the value obtained is consistent, standard deviation of each algorithms turnaround time of each set of experiments, based on 128 procesors executing 100000 to 1000000 variations of jobs were computed. Table 4.28, showed the standard deviation of Figure 4.28 results.

Table 4.28: Standard Deviation

Standard Deviation	FCFS	RR	EDF	EDFRR	EDFRRGCD	EDFRRLCM	LST	LSTRR	LSTRRGCD	LSTRRLCM	LSTRF	LSTRFRR	LSTRFRRGCD	LSTRFRRLCM
100000	5.73E+08	3.71E+08	2.64E+08	3.75E+08	3.78E+08	3.59E+08	2.07E+08	3.74E+08	3.59E+08	2.11E+08	1.90E+08	3.74E+08	3.54E+08	1.93E+08
200000	9.55E+08	5.67E+08	3.70E+08	4.77E+08	5.22E+08	5.64E+08	3.20E+08	5.72E+08	4.82E+08	3.24E+08	2.89E+08	5.72E+08	4.73E+08	2.94E+08
300000	2.74E+09	2.80E+09	1.38E+09	1.84E+09	2.01E+09	1.64E+09	1.30E+09	1.84E+09	1.68E+09	1.08E+09	1.21E+09	1.84E+09	1.66E+09	9.91E+08
400000	3.90E+09	2.60E+09	1.82E+09	2.61E+09	2.53E+09	2.57E+09	1.65E+09	2.61E+09	2.41E+09	1.51E+09	1.53E+09	2.61E+09	2.28E+09	1.39E+09
500000	5.53E+09	3.26E+09	2.48E+09	3.27E+09	3.28E+09	3.43E+09	2.25E+09	3.27E+09	3.04E+09	1.83E+09	2.02E+09	3.27E+09	2.99E+09	1.63E+09
600000	5.53E+09	3.51E+09	2.66E+09	3.52E+09	3.55E+09	3.83E+09	2.51E+09	3.52E+09	3.29E+09	1.93E+09	2.23E+09	3.52E+09	3.23E+09	1.65E+09
700000	8.04E+09	4.40E+09	3.11E+09	4.42E+09	4.45E+09	4.91E+09	2.83E+09	4.41E+09	4.14E+09	2.46E+09	2.59E+09	4.41E+09	4.07E+09	2.15E+09
800000	8.28E+09	4.83E+09	3.57E+09	4.84E+09	4.95E+09	5.68E+09	3.15E+09	4.84E+09	4.56E+09	2.73E+09	2.79E+09	4.84E+09	4.48E+09	2.35E+09
900000	1.03E+10	5.54E+09	4.14E+09	5.55E+09	5.67E+09	6.85E+09	3.62E+09	5.55E+09	5.19E+09	3.25E+09	3.13E+09	5.55E+09	5.10E+09	2.79E+09
1000000	1.12E+10	5.95E+09	4.28E+09	5.97E+09	5.98E+09	3.86E+10	3.80E+09	5.97E+09	5.97E+09	3.57E+09	3.25E+09	5.97E+09	5.97E+09	3.53E+09

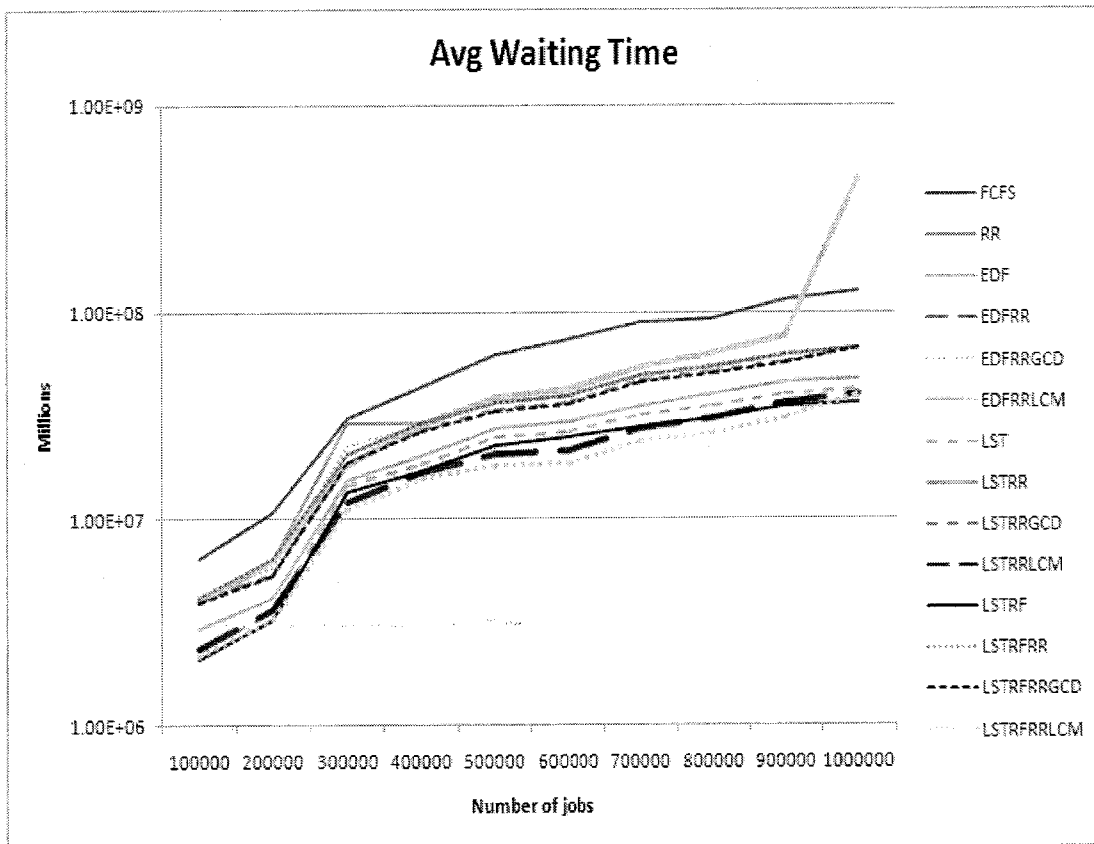


Figure 4.29: Average Waiting Time

It can be observed from Figure 4.29 that LST, RR, LSTRR, LSTRRGCD, EDFRRGCD, LSTRFRR, LSTRFRRGCD, LSTRFRRLCM, and LSTRRLCM showed sharp rise from 100000 to 200000 number of jobs, from 200000 to 300000, then a slight rise from 300000 to 500000 number of jobs, then a slight fall from 500000 to 600000 number of jobs, then a slight rise from 600000 to 700000 number

of jobs, finally smooth and steady from 700000 to 1000000 number of jobs. LSTRF showed sharp rise from 100000 to 2000000 number of jobs, from 200000 to 300000, then a slight rise from 300000 to 500000 number of jobs, then finally smooth and steady from 500000 to 1000000 number of jobs. EDFRR showed sharp rise from 100000 to 2000000 number of jobs, from 200000 to 300000, a slight fall from 300000 to 400000 number of jobs, a slight rise from 400000 to 500000 number of jobs, a slight fall from 500000 to 600000 number of jobs, then a slight rise from 600000 to 700000 number of jobs, finally smooth and steady from 700000 to 1000000 number of jobs. EDFRRLCM showed sharp rise from 100000 to 2000000 number of jobs, from 200000 to 300000, then a slight rise from 300000 to 500000 number of jobs, then a slight fall from 500000 to 600000 number of jobs, then a slight rise from 600000 to 700000 number of jobs, then smooth and steady from 700000 to 900000 number of jobs, then finally a sharp rise from 700000 to 1000000 number of jobs. Meanwhile, FCFS showed sharp rise from 100000 to 2000000 number of jobs, from 200000 to 300000, then a slight rise from 300000 to 500000 number of jobs, then a slight rise from 500000 to 700000 number of jobs, then a slight fall from 700000 to 800000 number of jobs, then a slight fall from 800000 to 900000 number of jobs, then finally slight rise from 900000 to 1000000 number of jobs.

Moreover, RR, LSTRR, LSTRRGCD, EDFRR, EDFRRGCD, LSTRFRR, LSTRFRRGCD, LSTRFRRLCM and LSTRRLCM are overlapping one another. But there is a slight difference between the algorithms. Results showed that LSTRFRRLCM and LSTRRLCM have the best performance, while FCFS and EDFRRLCM showed the worst performance.

To ensure that the value obtained is consistent, standard deviation of each algorithms turnaround time of each set of experiments, based on 128 procesors executing 100000 to 1000000 variations of jobs were computed. Table 4.29, showed the standard deviation of Figure 4.29 results.

Table 4.29: Standard Deviation

Standard Deviation	FCFS	RR	EDF	EDFRR	EDFRRGCD	EDFRRLCM	LST	LSTRR	LSTRRGCD	LSTRRLCM	LSTRF	LSTRFR	LSTRFRGCD	LSTRFRRLCM
100000	5.72E+08	3.70E+08	2.63E+08	3.73E+08	3.77E+08	3.58E+08	2.06E+08	3.73E+08	3.58E+08	2.09E+08	1.89E+08	3.73E+08	3.53E+08	1.92E+08
200000	9.54E+08	5.68E+08	3.69E+08	4.77E+08	5.21E+08	5.63E+08	3.19E+08	5.71E+08	4.00E+08	3.23E+08	2.68E+08	5.70E+08	4.72E+08	2.99E+08
300000	2.74E+09	2.60E+09	1.38E+09	1.84E+09	2.01E+09	1.64E+09	1.30E+09	1.84E+09	1.68E+09	1.07E+09	1.21E+09	1.84E+09	1.66E+09	9.88E+08
400000	3.90E+09	2.60E+09	1.82E+09	2.51E+09	2.53E+09	2.57E+09	1.65E+09	2.61E+09	2.41E+09	1.50E+09	1.53E+09	2.61E+09	2.38E+09	1.39E+09
500000	5.53E+09	3.28E+09	2.47E+09	3.27E+09	3.27E+09	3.43E+09	2.24E+09	3.27E+09	3.04E+09	1.83E+09	2.02E+09	3.27E+09	2.99E+09	1.63E+09
600000	6.62E+09	3.51E+09	2.66E+09	3.52E+09	3.54E+09	3.83E+09	2.61E+09	3.52E+09	3.28E+09	1.92E+09	2.23E+09	3.52E+09	3.23E+09	1.66E+09
700000	8.03E+09	4.40E+09	3.11E+09	4.41E+09	4.45E+09	4.90E+09	2.83E+09	4.41E+09	4.14E+09	2.46E+09	2.50E+09	4.41E+09	4.07E+09	2.14E+09
800000	8.29E+09	4.82E+09	3.57E+09	4.84E+09	4.95E+09	5.67E+09	3.15E+09	4.84E+09	4.56E+09	2.72E+09	2.78E+09	4.84E+09	4.47E+09	2.35E+09
900000	1.03E+10	5.54E+09	4.13E+09	5.55E+09	5.67E+09	6.85E+09	3.62E+09	5.55E+09	5.18E+09	3.25E+09	3.13E+09	5.55E+09	5.09E+09	2.79E+09
1000000	1.12E+10	5.95E+09	4.28E+09	5.97E+09	5.97E+09	3.86E+10	3.79E+09	5.96E+09	5.97E+09	3.57E+09	3.28E+09	5.96E+09	5.97E+09	3.53E+09

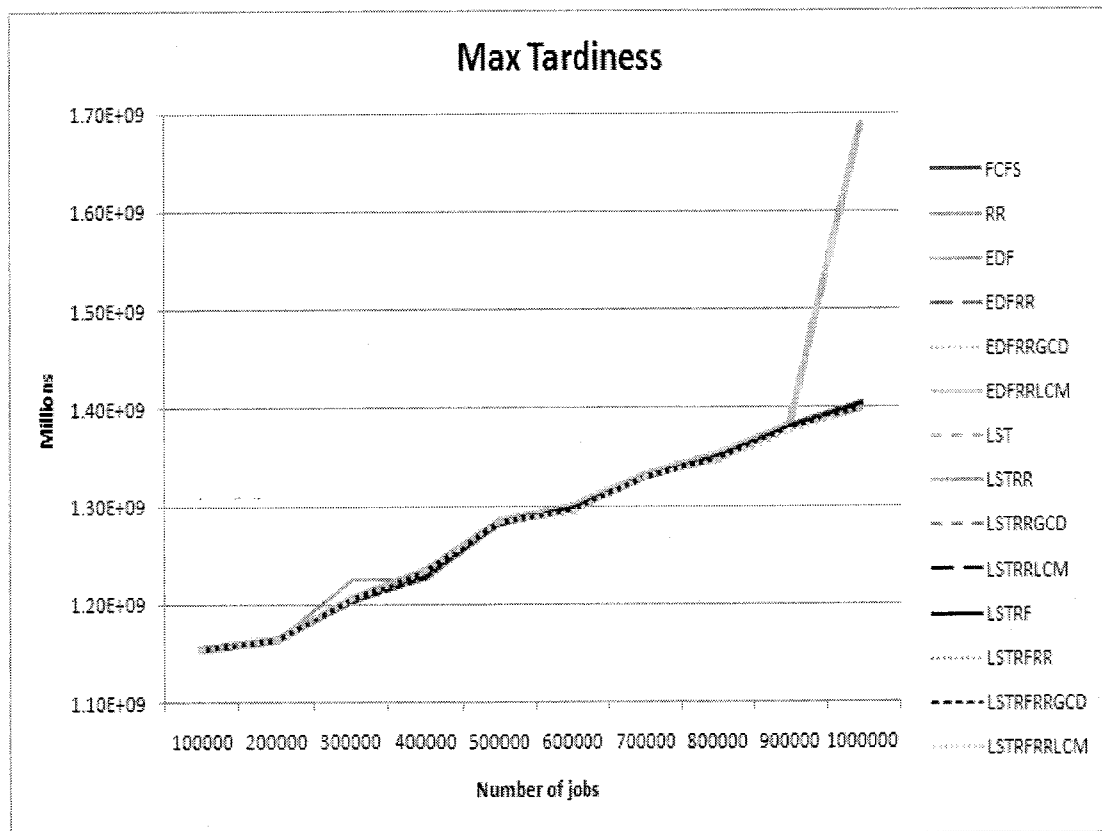


Figure 4.30: Maximum Tardiness

Figure 4.30 shows that maximum tardiness is not fixed it varies on the workload. However, it can be observed that LSTRF, LST, RR, LSTRR, LSTRRGCD,

EDFRRGCD, LSTRFRR, LSTRFRRGCD, FCFS EDF, LSTRFRRLCM, LSTRRLCM and EDFRRLCM, showed sharp rise from 100000 to 200000 numbers of jobs, from 200000 to 400000 numbers of jobs, from 400000 to 500000 numbers of jobs, then a slight fall from 500000 to 600000 numbers of jobs, then a slight rise from 600000 to 800000 numbers of jobs, then finally smooth from 800000 to 1000000 numbers of jobs. EDFRR showed sharp rise from 100000 to 200000 numbers of jobs, then a sharp rise from 200000 to 300000 numbers of jobs, then a slight fall from 300000 to 400000 numbers of jobs, a slight rise from 400000 to 500000 numbers of jobs, then a slight fall from 500000 to 600000 numbers of jobs, then a slight rise from 600000 to 800000 numbers of jobs, then finally smooth from 800000 to 1000000 numbers of jobs. Meanwhile, EDFRRLCM showed sharp rise from 100000 to 200000 numbers of jobs, from 200000 to 400000 numbers of jobs, from 400000 to 500000 numbers of jobs, then a slight fall from 500000 to 600000 numbers of jobs, then a slight rise from 600000 to 800000 numbers of jobs, then a slight rise from 800000 to 900000 numbers of jobs, then a sharp rise from 900000 to 1000000 numbers of jobs. Results showed that LSTRFLCM and LSTRF have the best performance, while LSTRFRRGCD and EDFRRLCM showed the worst performance. To ensure that the value obtained is consistent, standard deviation of each algorithms turnaround time of each set of experiments, based on 128 processors executing 100000 to 1000000 variations of jobs were computed. Table 4.30, showed the standard deviation of Figure 4.30 results.

Table 4.30: Standard Deviation

Standard Deviation	FCFS	RR	EDF	EDFRR	EDFRRGCD	EDFRRRLCM	LST	LSTRR	LSTRRGCD	LSTRRLCM	LSTRF	LSTRFRR	LSTRFRRGCD	LSTRFRRLCM
100000	1.04E+11	1.04E+11	1.04E+11	1.04E+11	1.04E+11	1.04E+11	1.04E+11	1.04E+11	1.04E+11	1.04E+11	1.04E+11	1.04E+11	1.04E+11	1.04E+11
200000	1.04E+11	1.04E+11	1.04E+11	1.04E+11	1.05E+11	1.05E+11	1.04E+11	1.05E+11	1.05E+11	1.05E+11	1.04E+11	1.05E+11	1.05E+11	1.05E+11
300000	1.08E+11	1.10E+11	1.08E+11	1.08E+11	1.08E+11	1.08E+11	1.08E+11	1.08E+11	1.08E+11	1.08E+11	1.08E+11	1.08E+11	1.08E+11	1.08E+11
400000	1.11E+11	1.10E+11	1.11E+11	1.10E+11	1.10E+11	1.11E+11	1.11E+11	1.10E+11	1.10E+11	1.11E+11	1.11E+11	1.10E+11	1.10E+11	1.11E+11
500000	1.15E+11	1.15E+11	1.15E+11	1.15E+11	1.15E+11	1.15E+11	1.15E+11	1.28E+09	1.15E+11	1.15E+11	1.15E+11	1.15E+11	1.15E+11	1.15E+11
600000	1.17E+11	1.18E+11	1.16E+11	1.17E+11	1.17E+11	1.17E+11	1.18E+11	1.17E+11	1.17E+11	1.18E+11	1.18E+11	1.17E+11	1.17E+11	1.16E+11
700000	1.19E+11	1.19E+11	1.19E+11	1.19E+11	1.19E+11	1.19E+11	1.19E+11	1.19E+11	1.19E+11	1.19E+11	1.19E+11	1.19E+11	1.19E+11	1.19E+11
800000	1.21E+11	1.21E+11	1.21E+11	1.21E+11	1.21E+11	1.21E+11	1.21E+11	1.21E+11	1.21E+11	1.21E+11	1.21E+11	1.21E+11	1.21E+11	1.21E+11
900000	1.24E+11	1.24E+11	1.24E+11	1.24E+11	1.24E+11	1.24E+11	1.24E+11	1.24E+11	1.24E+11	1.24E+11	1.24E+11	1.24E+11	1.24E+11	1.24E+11
1000000	1.26E+11	1.26E+11	1.26E+11	1.26E+11	1.26E+11	1.26E+11	1.26E+11	1.26E+11	1.26E+11	1.26E+11	1.26E+11	1.26E+11	1.26E+11	1.26E+11

CHAPTER 5

CONCLUSION AND FUTURE RESEARCH

5.1 Chapter Overview

The chapter summarizes the major findings of this research, beginning from literature review, algorithms development and benchmarking.

5.2 Conclusion

This research had dual goals: to integrate new robust hybrid methods based on baseline approaches and to implement, evaluate and test these developed algorithms with real benchmark traces on real computational grid environment. These two objectives have been realized.

5.2.1 Outcome of the Literature Review

The literature review showed that though much work has been done on grid scheduling, not much report is found dealing with scheduling algorithms that combined deadline and slack time in their development. In the same vein, the presented algorithms were not implemented using hard- and soft- real-time system approach.

5.2.2 Outcome of Algorithm Development

Fourteen (14) algorithms were developed. Of the fourteen, five (5) are baseline approaches, while the rest are hybrids based on real-time system and round robin fairness, using operational research (OR) concepts.

Extensive performance analyses were carried out using real workload traces in real computational grid environment to evaluate the efficiency and robustness of the developed grid scheduling algorithms with respect to the following performance metrics: average turnaround time average waiting time and maximum tardiness.

As expected, the time required to perform scheduling in the dynamic situation becomes stable when varying the number of jobs and processors under an increasing real workload. However, it is clear that the performance metrics (average turnaround time, average waiting time and maximum tardiness) strongly dependent on the number of available processors. Their performance becomes higher when the numbers of processors are increased.

Based on the comparative performance analysis between the developed scheduling algorithms and the baseline approaches (the chosen benchmarks), results have shown that LSTRF, LSTRFRRLCM and LST scheduling algorithms have the best performance among all the compared scheduling algorithms (FCFS, RR, EDF, LSTRR, LSTRFRR, LSTRRGCD, LSTRFRRGCD, EDFRRGCD, LSTRRLCM, and EDFRRLCM), while EDFRRLCM and FCFS showed the worst performance. Therefore, we conclude that LSTRF, LSTRFRRLCM and LST scheduling approaches, which could be used in solving real grid computational challenges and could be made as part of the general grid scheduling solution policy. This conclusion stems from the fact that the real grid infrastructure requirements (Short Turnaround Time, Short Average Waiting Time and Short Tardiness Time) as well as the maintenance of scalability under heavy workload and varying number of processors in a real computational grid environment are adequately met by these developed algorithms.

Thus, combining real-time system and round robin fairness (soft real-time system) techniques has maximized the number of priority tasks in meeting their deadlines in addition to creating fairness between the tasks and processors.

5.3 Research Limitation

Based on our research we cannot draw any conclusion of the best algorithms on the real distributed environment. What will be the best performing algorithms with respect to green parameters are yet to be considered.

5.4 Recommendations and Future Works

Further exploration of the proposed scheduling techniques should be done in future. Future work includes the following:

- Refinement of the existing algorithms.
- Developing some new algorithms may be based on some evolutionary approaches like Genetic Algorithm, Simulated Annealing or Ant Optimization.
- We will highlight incorporation of the green parameters with the existing algorithms.
- Testing the proposed scheduling model in true heterogeneous distributed environment can also be done in order to evaluate its performance in true grid environment.

REFERENCES

- [1] I. Foster, "What is the Grid? A Three Point Checklist," *Argonne National Laboratory and University of Chicago*, 2002.
- [2] I. Foster and C. Kesselman, "The Grid: Blueprint for a Future Computing Infrastructure," *Morgan Kaufmann*, 1998.
- [3] A. Abbas, "Grid Computing: A Practical Guide to Technology and Applications," *Charles River Media*, 2004.
- [4] F. Berman, F. Geoffrey, and J. G. Anthony, "Grid Computing: Making The Global Infrastructure a Reality," *Wiley Series in Communications, Networking and Distributed System*, 2003.
- [5] M. Baker, R. Buyya, and D. Laforenza, "Grids and grid technologies for wide-area distributed computing," *International Journal of Software: Practice and Experience (SPE)*, December 2002
- [6] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations.," *International Journal of Supercomputing Applications*, , vol. 15 (3). 2001. .
- [7] A. Bar-Noy, M. Halldorsson, H. Shachnai, and T. Tamir, "On chromatic sums and distributed resource allocation," *Information and Computation*, vol. 140, pp. 183–202, 1998.
- [8] I. Foster and C. Kesselman, "Grid: Blueprint for a New Computing Infrastructure," *Morgan Kaufmann*, 1999.
- [9] L. Zhang, J. Chung, and Q. Zhou, "Developing Grid computing applications," *Discover Grid computing, developerWorks Journal*, vol. 14-19, February 2003.

- [10] G. _Alliance, "USC/Information Sciences Institute 4676 Admiralty Way, Suite 1001," *Press Releases, c/o Carl Kesselman*.
- [11] A. L. Pereira, V. Muppavarapu, and S. M. Chung, "Role-Based Access Control for Grid Database Services Using the Community Authorization Service," *IEEE Trans. on Dependable and Secure Computing*, vol. 3, pp. 156-166, 2006.
- [12] G. Aram, K. Czajkowski, and K. Lerman, "Resource allocation in the grid with learning agents," *J. Grid Comput.*, vol. 3, pp. 91-100, 2005.
- [13] R. Buyya, D. Abramson, and J. Giddy, "A Case for Economy Grid Architecture for Service-Oriented Grid Computing," *Proceedings of the International Parallel and Distributed Processing Symposium: 10th IEEE International Heterogeneous Computing Workshop (HCW 2001), April 23, 2001, San Francisco, California, USA, IEEE CS Press, USA., 2001*.
- [14] T. A. A. Project., December 2004.
- [15] V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour, "Evaluation of Job-Scheduling Strategies for Grid Computing. ," *In 1st IEEE/ACM International Workshop on Grid Computing (Grid 2000), Berlin, Lecture Notes in Computer Science (LNCS), Springer-Verlag, Heidelberg, Germany, , pp. 191-202. , 2000*.
- [16] G. Mateescu, "Quality of Service on the Grid via Metascheduling with Resource Co-scheduling and Co-reservation. ," *International Journal of High Performance Computing Applications, SAGE Publications Inc, London, UK. , vol. 17, pp. 209-218, 2003*.
- [17] B. Wei and D. Zhang, "A Novel Least Slack First Scheduling algorithm Optimized by Threshold," *Control Conference, CCC, Chinese*, pp. 264 - 268 2007.

- [18] H. Myungwon, C. Dongjin, and K. PanKoo, "Least Slack Time Rate first: New Scheduling Algorithm for Multi Processor Environment," *International Conference on Complex, Intelligent and Software Intensive Systems.*, 2010
- [19] E. M. Goldratt "Theory of Constraints," *Great Barrington, MA : North River Press operations research bottleneck and non-bottleneck work centers.*, 1999.
- [20] G. Tibor, "A Resource Allocation Protocol for Providing Quality of Service in Grid Computing, using a Policy-Based Approach," *AICT-ICIW '06 Proceedings of the Advanced Int'l Conference on Telecommunications and Int'l Conference on Internet and Web Applications and Services, IEEE Computer Society Washington, DC, USA*, 2006.
- [21] K. Somasundaram and S. Radhakrishnan, "'Task Resource Allocation in Grid using Swift Scheduler'," vol. IV, No. 2., pp. 158-166., 2009.
- [22] A. Sulistio, U. Cibej, S. Venugopal, B. Robic, and R. Buyya, "A Toolkit for Modelling and Simulating Data Grids: An Extension to GridSim. Concurrency and Computation," *Practice and Experience (CCPE), Wiley Press, New York, USA*, 2008.
- [23] I. Leila, "Dynamic Resource Allocation Mechanisms for Grid Computing," *IEEE International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities 2007*.
- [24] R. H. Gamma-Erich, J. Ralph, and V. John, "Design Patterns," *Addison-Wesley Professional, Pts Edn*, 1995.
- [25] L. Jun, G. Andre, and R. Catherine, "Efficient Algorithms to Solve a Class of Resource Allocation Problems in Large Wireless Networks. ," *WiOPT'09 Proceedings of the 7th International Conference on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks IEEE Press Piscataway, NJ, USA*, 2009.
- [26] X. L. Du, C. J. Jiang, G. R. Xu, and Z. J. Ding, "A Grid DAG scheduling algorithm based on fuzzy clustering.," *J. Software*, vol. 17, 2006.

- [27] M. Dorigo and C. Blum, "Ant Colony Optimization Theory: A Survey," *Theor. Comput. Sci.*, , vol. 344: 243- 278, 2005.
- [28] R. Buyya and M. Murshed, "GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing.," *Concurr. Comput. Praticice Exper. (CCPE)*, , vol. 14, p. 1175, 2002.
- [29] C. Chun-Tian and L. Zhi-Jie, "Parallel algorithm for grid resource allocation based on nash equilibrium. ," *Computat.*,, vol. 1, pp. 53-66, 2006.
- [30] R. Buyya, "Economic-based distributed resource management and scheduling for grid computing, (Ph.D. dissertation), ," *Melbourne, Australia: Monash University*, 2002.
- [31] Subramoniam.K., M. Maheswaran, and M. Toulouse, "Towards a micro economic model for resource allocation in grid computing systems.," *Proceedings of the 2002 IEEE Canadian Conf. on Electrical and Computer Engineering, Manitoba, Canada.*, 2002.
- [32] M. L. Weng, L. X. Chuliang, and Q. .D, "An Economic-based Resource Management Framework in the Grid context.," *Proceedings of the 5th IEEE International Symposium on Cluster Computing and the Grid (CCG), Cardiff, Wales, UK.*, 2005.
- [33] L. FuFang and Q. DeYu, "Research on Grid Resource Allocation Algorithm Based on Fuzzy Clustering. Future Generation Communication and Networking, 2008," *FGCN '08. 2nd International Conference*, pp. 162-166., 2008.
- [34] Z. Ru-Huai, "The Net-Masking for the Fuzzy Clustering.," *J. Xi'An JiaoTong University.*,, vol. 14, pp. 29-36, 1980.
- [35] S. Dawei, C. Guiran, J. Lizhong, and W. Xingwei, "Optimizing Grid Resource Allocation by Combining Fuzzy Clustering with Application Preference" " *Control (ICACC), 2010 2nd International Conference*, pp. 22-27. , 2010.

- [36] D. F. Ferguson, C. Nikolaou, J. Sairamesh, and Y. Yemini, "Economic Models for Allocating Resources in Computer Systems. Resource Allocation, ," *In Scott Clearwater, editor, Market-Based Control: A Paradigm for Distributed Resource Allocation, World Scientific, Hong Kong., 1996.*
- [37] C. Li and L. Li, "A Utility-based Two Level Market Solution For Optimal Resource Allocation In Computational Grid.," *Parallel Processing, 2005. ICPP 2005. International Conference on, , pp. 23-30., 2005.*
- [38] R. Buyya, J. Giddy, and D. Abramson, "A Case for Economy Grid Architecture for Service-Oriented Grid Computing " *10th IEEE International Heterogeneous Computing Workshop (HCW 2001), In conjunction with IPDPS 2001, San Francisco, California, USA, April., 2001.*
- [39] Y. Adil, A. Abdul-Hanan, and A. A. Aboamama, "A bidding-based grid resource selection algorithm using single reservation mechanism.," *Int. J. Comp. Appl., , vol. 16, 2011..*
- [40] J. Cao, S. A. Jarvis, S. Saini, and G. R. Nudd, "GridFlow:Workflow Management for Grid Computing.," *In 3rd International Symposium on Cluster Computing and the Grid (CCGrid), Tokyo, Japan, IEEE Computer Society Press, Los Alamitos., 2003.*
- [41] S. Lorpunmanee, M. N. M. Sap, and A. Abdul-Hanan, " fuzzy c-mean and genetic algorithms based scheduling for Independent jobs in computational grid," *Jurnal Teknologi Maklumat, vol. 18, 2006.*
- [42] S. Lorpunmanee, M. N. M. Sap, and A. Abdul-Hanan, "Optimalisation of a Job Scheduler in the Grid Environment by Using Fuzzy C-Mean" " *J. J. Appl. Sci., , vol. 9, 2007.*
- [43] P. Florin, T. Dacian, C. Valentin, and C. Vladimir, "Fault-Tolerant Scheduling Framework for MedioGRID System," *EUROCON 2007 The International Conference on "Computer as a Tool" IEEE , Warsaw, September 9-12, vol. 1-4244-0813 2007.*

- [44] K. Snehal and D. Neeta, "Efficient CPU Scheduling: A Genetic Algorithm based Approach," *Ad Hoc and Ubiquitous Computing, 2006. ISAUHC '06. International Symposium On -1/06/©2006IEEE.*, vol. 1-4244-0731, pp. 206 – 207, 2006.
- [45] A. Bouyer, B. Arasteh, and A. Movaghar, "A new Hybrid Model using Case-Based Reasoning and Decision Tree Methods for improving Speedup and Accuracy," *Iadis International Conference Applied Computing 2007*, .
- [46] R. Ivan, G. Francese, and C. Julita, "Evaluation of Coordinated Grid Scheduling Strategies," *High Performance Computing and Communications, HPCC '09. 11th IEEE International Conference* pp. 1 – 10, 2009.
- [47] Y.-P. Bu, W. Zhou, and J.-S. Yu, "An Improved PSO Algorithm and Its Application to Grid Scheduling Problem," *International Symposium on Computer Science and Computational Technology, -5/08 © 2008 IEEE*, 2008
- [48] P. Mathiyalagan, U. R. Dhephthie, and S. N. Sivanandam, "Grid scheduling using Enhanced PSO algorithm," *P.Mathiyalagan et al. / (IJCSSE) International Journal on Computer Science and Engineering* ,, vol. 02, pp. 140-145, 2010, .
- [49] Z. Pooranian, A. Harounabadi, M. Shojafar, and J. Mirabedini, "Hybrid PSO for Independent Task scheduling in Grid Computing to Decrease Makespan," *International Conference on Future Information Technology IPCSIT*,, vol. 13 2011.
- [50] V. S. SharmaKant, M. K. Mishra, P. P. Bhuyan, and U. C. Dey, "An Agent Based Dynamic Resource Scheduling Model with FCFS-Job Grouping Strategy in Grid Computing," *World Academy of Science, Engineering and Tech*, 2010.
- [51] B. Asgarali and M. N. SAP, "A Prediction-based Fault Tolerance on Grid Resources scheduling by using Optimized Case-based Reasoning,"

Proceedings, Faculty of computer science and information systems University technology of Malaysia., 2009.

- [52] M. Hiroyuki, H. Masahiro, I. Daisuke, A. Yutaka, and Y. Naoaki, "Advanced Wavelength Reservation Method Based on Deadline-Aware Scheduling for Lambda Grid Networks," *Journal of lightwave technology.*, vol. 25, October 2007.
- [53] R. Buyya, M. Murshed, D. Abramson, and S. Venugopal, "Scheduling parameter sweep applications on global Grids: a deadline and budget constrained cost-time optimization algorithm," *software practice and experience* vol. 35, pp. 491-512, 2005.
- [54] S. B. Liu and L. Shuang, "A General Distributed Scalable Peer to Peer Scheduler for Mixed Tasks in Grids," *Springer-Verlag Berlin Heidelberg* pp. pp. 320-330, 2007.
- [55] C. Eddy, K. C. Pushpinder, and D. Frederic, "Deadline Scheduling with Priority for Client-Server Systems on the Grid," *Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04)* 2004.
- [56] A. Takefusa, H. Casanova, S. Matsuoka, and F. Berman, "A study of deadline scheduling for client-server systems on the computational grid.," *In the 10th IEEE Symposium on High Performance and Distributed Computing (HPDC'01), San Francisco, California., 2001.*
- [57] A. Ballier and C. Eddy, "Simulating Grid Schedulers with Deadlines and Co-Allocation," *FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002 004265). Project no. FP6-004265., 2002.*
- [58] D. Klusacek and H. Rudova, "Improving QoS in computational Grids through schedule-based approach. In Scheduling and Planning Applications Workshop " *Eighteenth International Conference on Automated Planning and Scheduling (ICAPS'08), Sydney, Australia., 2008.*

- [59] A. Aggarwal, P. Du, and R. D. Kent, "Grid Scheduling Optimization Based on Resource Characteristics," *Journal of Computational Information Systems*, vol. 6, pp. 4609-4616, December, 2010.
- [60] L.-Y. Tseng, C. Yeh-Hao, and W. Shu-Ching, "A Deadline-Based Task Scheduling With Minimized Makespan," *International Journal of Innovative Computing, Information and Control* vol. 5, pp. 1665-1679, June 2009
- [61] S. Wang, X. Yun, and X. Yu, "Survivability-based Scheduling Algorithm for Bag-of-Tasks Applications with Deadline Constraints on Grids," *International Journal of Computer Science and Network Security.*, vol. 6, April 2006.
- [62] L. Cong and S. Baskiyar, "Scheduling Mixed Tasks with Deadlines in Grids Using Bin Packing," *Parallel and Distributed Systems. ICPADS '08. 14th IEEE International Conference* p. 229, 2008.
- [63] D. P. Spooner, S. A. Jarvis, J. Caoy, S. Saini, and G. R. Nudd, "Local Grid Scheduling Techniques using Performance Prediction," *Computers and Digital Techniques, IEE Proceedings* vol. 150 pp. 87 - 96, Mar 2003.
- [64] D. Fang, L. Junzhou, G. Lisha, and G. Liang, "A Grid Task Scheduling Algorithm Based on QoS Priority Grouping," *Proceedings of the Fifth International Conference on Grid and Cooperative Computing (GCC'06) IEEE*, 2006.
- [65] M. Hiroyuki, H. Masahiro, I. Daisuke, A. Yutaka, and Y. Naoaki, "A Deadline-Aware Wavelength Scheduling scheme for WDM-based, Grid Networks," *IEEE International Conference ICC '07*, p. 2383, June 2007. .
- [66] V. Sundaram, A. Chandra, and J. Weissman, "Exploring the Throughput-Fairness Tradeoff of Deadline Scheduling in Heterogeneous Computing Environments," *Proceeding SIGMETRICS '08 Proceedings of the ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pp. 463-464 2008.

- [67] S. K. Garg, R. Buyya, and C. J. Siegel, "Scheduling Parallel Applications on Utility Grids: Time and Cost Trade-Off Management," *Thirty-Second Australasian Computer Science Conference, Conferences in Research and Practice in Information Technology (CRPIT)*, vol. 91, 2009.
- [68] K. Dalibor and R. Hana, "Comparison Of Multi-Criteria Scheduling Techniques," *In CoreGRID Integration Workshop. Integrated Research in Grid Computing. Heraklion-Crete*, 2008.
- [69] G. K. Kamalam and B. V. Murali, "An Efficient Hybrid Job Scheduling Algorithm for Computational Grids," *International Conference on Web Services Computing (ICWSC) Proceedings published by International Journal of Computer Applications®(IJCA)*. 2011.
- [70] J. Y. Hongwei Liu, Guozhong Tian, and Hongcui Gong, "The Priority Tasks Scheduling Algorithm Based on Grid Resource Prediction," *Fourth ChinaGrid Annual Conference* pp. 84 -87, 2009.
- [71] B. Li and B. Shen, "Slack-based Advance Reservation for Grid Jobs," *Advanced Computer Theory and Engineering (ICACTE), 3rd International Conference*, vol. V3-418 - V3-421, 2010.
- [72] S. Behera, "An improved least-laxity-first Scheduling algorithm for Real-time tasks," *International Journal of Engineering Science and Technology (IJEST)*, vol. 4, April 2012.
- [73] G.-F. Golnar, M.-d. Fahime, D. Hossein, and M. Anahita, "Scheduling of scientific workflows using a chaos-genetic algorithm," *International Conference on Computational Science, ICCS.*, 2010.
- [74] C. Javier and A. Unai, "Distributed Scheduler of Workflows with Deadlines in a P2P Desktop Grid," *Parallel, Distributed and Network-Based Processing (PDP), 18th Euromicro International Conference*, pp. 69 – 73, 2010
- [75] S. Viswanathan, B. Veeravalli, and T. G. Robertazzi, "Resource-Aware Distributed Scheduling Strategies for Large-Scale Computational Cluster/Grid

- Systems," *IEEE Transactions On Parallel And Distributed Systems*, OCTOBER., vol. 18, 2007.
- [76] D. D. Nikolaos, D. D. Anastasios, A. V. Emmanouel, and A. V. Theodora, "Fair Scheduling Algorithms in Grids," *IEEE Transactions On Parallel And Distributed Systems*, NOVEMBER., vol. 18, 2007.
- [77] Y. Jia, B. Rajkumar, and K. T. Chen, "Cost-based Scheduling of Scientific Workflow Applications on Utility Grids," *Proceedings of the First International Conference on e-Science and Grid Computing* vol. 0-7695-2448, 2005.
- [78] A. Sulistio, U. Cibej, S. Venugopal, B. Robic, and R. Buyya, "A Toolkit for Modelling and Simulating Data Grids: An Extension to GridSim," *Concurrency and Computation: Practice and Experience (CCPE)*, Online ISSN: 1532-0634, Printed ISSN: 1532-0626, 20(13): 1591-1609, Wiley Press, New York, USA, Sep. , 2008.
- [79] L. Daphne and R. S. V. Kasmir, "A Dynamic Error Based Fair Scheduling Algorithm For A Computational Grid," *Journal of Theoretical and Applied Information Technology*, 2005 -2009.
- [80] "Grid Workloads Archive (<http://gwa.ewi.tudelft.nl/pmwiki/>)."
- [81] B. D. Henri, "DAS-2 team (http://gwa.ewi.tudelft.nl/pmwiki/reports/gwa-t-1/trace_analysis_report.html)."
- [82] C. Franck, "Grid'5000 team (<http://www.grid5000.org/>)."
- [83] K. Balasz, "The NorduGrid team (<http://www.nordugrid.org/>)."
- [84] M. Emmanuel, "The AuverGrid team (<http://www.auvergrid.fr/>)."
- [85] M. John, "SHARCNET, <john_x_sharcnet.ca> (<https://www.sharcnet.ca/my/front/>)."
- [86] I. C. London, "HEP e-Science group (<http://lcg.web.cern.ch/LCG/>)."

- [87] H. L. Anderson, "Metropolis, Monte Carlo and the MANIAC," *Los Alamos Science* vol. 14: 96-108., 1986.

APPENDIX A
PUBLICATIONS

PUBLICATIONS

H. A. Abba, N. Zakaria, A.J. Pal, and Ken Naono, Performance Comparison of Some Hybrid Deadline Based Scheduling Algorithms for Computational Grid, *Advances in Information Technology, 5th International Conference, IAIT 2012, Bangkok, Thailand, December 6-7, 2012. Springer book chapter*

H. A. Abba, N. Zakaria, S. N. M. Shah and A.J. Pal, Design, Development And Performance Analysis Of Deadline Based Priority Heuristic For Job Scheduling On A Grid, *Elsevier Journal of Procedia Engineering, to be presented in International Conference on Advances Science and Contemporary Engineering 2012, 24-25 Oct 2012.*

H. A. Abba, N. Zakaria and S. N. M. Shah, Deadline Based Performance Evaluation of Job Scheduling Algorithms, *IEEE International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC 2012), to appear, Sanya, Hainan, China, Oct. 10-12, 2012*

H. A. Abba, N. Zakaria, and Nazleeni Haron, Grid Resource Allocation: A Review, *Research Journal of Information Technology, 4(2): 38-55. June 30, 2012.*

WORKSHOP

Desktop Grid Computing and Applications *ICCIS 2012 Workshop June 2012*

