Master Thesis
Computer Science
Thesis no: MCS-2008-21
June 2008

Comparative Analysis of Static Recovery Schemes for Distributed Computing

Rashid Husain
Syed Mohammad Husnain Kazmi

Department of
Interaction and System Design
School of Engineering
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden

This thesis is submitted to the Department of Interaction and System Design, School of Engineering at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Computer Science. The thesis is equivalent to 20 weeks of full time studies.

Contact Information:
Author(s):
Rashid Husain
E-mail: sh.rashid.ravian@gmail.com
Syed Muhammad Husnain Kazmi
E-mail: shusnainkazmi@gmail.com

External advisor(s):
Kamilla Klonowska
Email: kamilla.klonowska@bth.se
Department of interaction and system Design
Blekinge institute of Technology, Sweden

Department of
Interaction and System Design
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden

Internet : www.bth.se/tek
Phone : +46 457 38 50 00
Fax : + 46 457 102 45

ABSTRACT

The primary objective of this thesis is to evaluate how grid computing works with their infrastructure. It also provides study to recognize the cases in which dynamic scheduler is preferable, how can the static recovery schemes play an effective role in large distributed system where load balancing is a key element and how can we get optimality in maximum number of crash computers using dynamic or static recovery schemes.

This thesis consists of two parts: construction of Golomb and Trapezium modules, and performance comparison of Golomb and Trapezium recovery schemes with dynamic recovery scheme.

In the first part we construct two modules that generate the recovery list of *n* computers, one for Golomb and one for Trapezium. In second part we make three schedulers, two for static recovery scheme and one for dynamic recovery scheme. In static recovery scheme we compare the performance of Golomb and Trapezium recovery scheme then we compare this performance with dynamic recovery scheme by using GridSim.

Keywords: Static recovery schemes, Golomb ruler, Trapezium recovery scheme, Dynamic Scheduler

Table of Contents

COMPARATIVE ANALYSIS OF STATIC RECOVERY SCHEMES FOR DISTRIBUTED COMPUTING I

ABSTRACT 1

PREFACE 4

1 INTRODUCTION 5

1.1 BACKGROUND 5
1.2 CHALLENGE/PROBLEM FOCUS 5
1.3 IMPORTANCE OF STUDY 5
1.4 GOAL/RESULT 5
1.5 METHOD/APPROACH 6
1.6 AUDIENCE 6
1.7 LIMITATION 6
1.8 MOTIVATION 6

2 BACKGROUND 7

2.1 GRID COMPUTING 7
2.2 CLUSTER COMPUTING 9

3 PROBLEM DEFINITION/GOAL 12

3.1 DYNAMIC LOAD BALANCING 12
3.1.1 Dynamic load balancing strategies 12
3.1.2 Taxonomy of dynamic load balancing 13
3.1.3 Advantages and disadvantages with Dynamic Load balancing 14
3.2 STATIC LOAD BALANCING 15
3.2.1 Advantages and disadvantages with static load balancing 15
3.3 SIMULATION TOOLKIT 16

4 METHODOLOGY 17

5 THEORETICAL WORK 18

5.1 DISTRIBUTED COMPUTING 18
5.1.1 Distributed system design framework 18
5.2 STATIC RECOVERY SCHEMES 19
5.2.1 Golomb Recovery scheme 20
5.2.2 Trapezium Recovery scheme 22

6 EMPIRICAL STUDY/CASE 24

6.1 SIMULATION TOOLS 24
6.2 GRIDSIM DESCRIPTION 25
6.2.1 GridSim entities 25
6.3 MAJOR SIMULATION RESULTS 26
6.3.1 Simulation Setup 26
6.3.2 Scenario with 7 computers, 01 Processing Element (PE) in Static recovery schemes 27
6.3.3 Scenario with 7 computers, 01 Processing Element (PE) in Dynamic Scheme 28
6.3.4 Scenario with 7 computers, 02 Processing Element (PE) in Dynamic Scheme 30
6.3.5 Comparison with static and Dynamic with different processing element 31
6.3.6 Analysis (Simulation results) 32

7 RESULTS 34

8 DISCUSSION/ANALYSIS 35

8.1 FUTURE WORK 36

SUMMARY **37**

REFERENCES **38**

TERMINOLOGY **41**

APPENDIX 1: OPTIMAL SEQUENCE **42**

APPENDIX 2: SEQUENCE OF GOLOMB RULERS **43**

APPENDIX 3: SEQUENCE OF TRAPEZIUM **44**

APPENDIX 4: DYNAMIC VS STATIC **45**

Table of Figures

Figure 2-1: Grid computing [1] **7**
Figure 2-2: Logical deployment levels of Grid computing [7] **8**
Figure 2-3: Grid topology [37] **9**
Figure 2-4: Architecture of HA OSCAR cluster [15] **10**
Figure 3-1: Taxonomy of Dynamic Load balancing [30] **13**
Figure 5-1: Distributed system design framework [27] **19**
Figure 5-2: Application executed on cluster with four computers [22] **20**
Figure 5-3: Optimal Golomb ruler (OGR) represented by differences [22] **21**
Figure 5-4: Triangle presentation of OGR [22] **21**
Figure 5-5: Triangle representation of the sequence S = < 1, 3, 2, 5, 2, 5> [23] **23**
Figure 6-1: Flow diagram in GridSim based simulations [4] **26**

PREFACE

It is difficult to mention the names of all those people without whom we would have not overcome the challenges we met in this work. However, the following people deserve to be particularly acknowledged.

We would like to express our utmost gratitude to our supervisor Dr. Kamilla Klonowska, for her contribution guidance, insightful ideas, comments, critical reviews, and encouragement during this work.

The simulation part of the work would have been difficult if it has not been assisted by Dr. Peter Tröger. Kashif Ahemd khan also gave good comments regarding simulation tools and in analysis. We would like to express our indebtedness to them.

I (Rashid Husain) want to express my gratitude to my family for their encouragement and support from afar during hard times. Special thanks to my friends Shahid Mahmood, Naveed Querashi, Bilal Raza, Faisal Hameed, Ijaz Butt, Jawad Ahmed, Ch Ahmed Nasir, Ahsan Elahi, Asim Upal, for their encouragement during that time.

Finally, I (Kazmi) would like to thank my family (especially my wife and mother) for their invaluable support, encouragement, understanding and patience during my busy times.

1 INTRODUCTION

1.1 Background

Availability is a key feature in the fault tolerant distributed systems, like clusters, and it can be achieved by using failover techniques [9]. For cluster availability, simplified strategies such as cold backup and warm backup are used by spare resources [32]. High cost can be a drawback for using many stand-by computers in a large cluster system in order to achieve predefined level of availability [9].

Another substitute solution for failure detection and recovery is the hot backup, even it is hard to make a decision on which computer the task of failure computer is executed while maintaining the load balance. Dynamic monitoring facilities and central scheduling are the usual solutions for this. In practice, the above solution turned out to be a problem with fault tolerant and scalability, while the mandatory scheduler acts both as a single point of failure and coordination bottleneck.

1.2 Challenge/problem focus

The following research questions are discussed in this thesis.

- What is state-of-the-art in static fault recovery schemes, especially under consideration of Golomb and Trapezium recovery schemes?
- How can simulation help in comparing the dynamic and new static fault tolerant recovery schemes?
- How do static recovery schemes perform in comparison to dynamic schemes?

1.3 Importance of study

The scientific analysis of alternatives to dynamic fault recovery is a relevant issue, since the dynamic approach has well-known limitations in terms of scalability, and those limitations have been discussed in chapter 3.1.3. The latest theoretical results in static recovery schemes [19] should therefore be compared to the classical approaches

1.4 Goal/Result

The objective of this thesis is to apply latest theoretical results in fault-tolerant load balancing to a modern distributed computing system. It concentrates on how recently discovered static recovery schemes can be applied to modern cluster and grid environments. The investigation is carried out by the help of simulation, including the comparison of Golomb and Trapezium recovery schemes with dynamic recovery scheme.

1.5 Method/Approach

We conducted a literature review to elucidate how fault tolerant parallel systems work with cluster and grid along with the functionality of classical schedulers, conventional fault tolerant recovery scheme and Golomb and Trapezium recovery schemes. Testing classical and new recovery schemes (Golomb and Trapezium) by simulations and their result analysis (best, equal, worst) are related to empirical investigations.

1.6 Audience

Our thesis is relevant to those people who have knowledge of Grid computing, Cluster, fault tolerance system and load balancing.

1.7 Limitation

This is possible to build large cluster and large grid environment, i.e. 1000 computers. However, the complexity of GridSim (the time of simulation) is so high that we had to limit the number of computers into 200. From the static point of view there is another limitation, i.e. an Optimal Golomb Ruler, which we explain in Chapter 5. In case of dynamic scenario we do not have to limit the number of computers.

Another limitation is the number of processors on each computer. The explanation here is that the previous theoretical results consists of this kind of problem. Also, with this scenario we can apply the results in a specific other systems, described in [21].

From this limitation we simply see the future work, when:

- We consider heterogonous computers, i.e. computers with different numbers of processors.
- Simulation with heterogonous networks, i.e. networks with different capacity of speed.

1.8 Motivation

A fundamental motivation for this research is that how distributed system work better in worst situation. As theoretically Kamilla klonowska et al has given this static recovery scheme (Golomb and Trapezium). In this thesis we have checked first time mathematical/theoretical results in real world environment with the help of Gridsim tools and it's very good for relevant user who wants to work in grid computing field

2 BACKGROUND

Humans, communication systems and computers are steadily being synergistically tied to make systems for sharing resources of all sorts with goals including scrutinizing and solving problems, implementing collaborations and sharing information. Internet, Grid based computation and sharing systems are examples of these types of systems [2].

“The terms "grid computing" and "cluster computing" have been used almost interchangeably to describe networked computers that run distributed applications and share resources”. Both technologies enable the shared use of distributed resources, and improve application performance by executing parallelizable computations concurrently on different machines [11].

In this chapter, we are going to describe the basic terminology like grid computing, grid topologies, cluster computing, high availability and fail over cluster, high performance computing (HPC), open source cluster application resources (OSCAR), and load balancing cluster.

2.1 Grid Computing

In [31], Grid Computing defines as:

“A parallel processing architecture in which CPU resources are shared across a network, and all machines function as one large supercomputer. It allows unused CPU capacity in all participating machines to be allocated to one application that is extremely computation intensive and programmed for parallel processing.”

Figure 2-1: Grid computing [1]

Grid computing is becoming increasingly vital in the area of the life sciences. Two particular aspects dominate in grid computing that is distributed computing and bioinformatics [40]. Distributed computing is typified by the fact that there exists a central management component that divides the program into smaller portions and distributes those to the connected computers. The central management component thus considers the heterogeneous resources when processing the allocation job [27]. Figure 2-1 shows that, Grid computing is all about sharing, aggregating hosting, and offering service across the globe for the advantages of human beings [1].

The most famous example of distributed computing is SETI@home developed by the University of California at Berkeley. It is basically a model of voluntary computing. SETI has become the pioneer in voluntary computing [27]. This kind of grid is mainly consisting of low powered computers with least application logic awareness and minimum storage capacity. The main benefits of computational grids are an abridged Total Cost of Ownership (TCO), and shorter deployment life cycles [**Error! Reference source not found.**].

Grid computing can be divided into three logical levels of deployment that shows in Figure 2-2, these are: [9]

- Global grids
- Enterprise grids
- Cluster grids

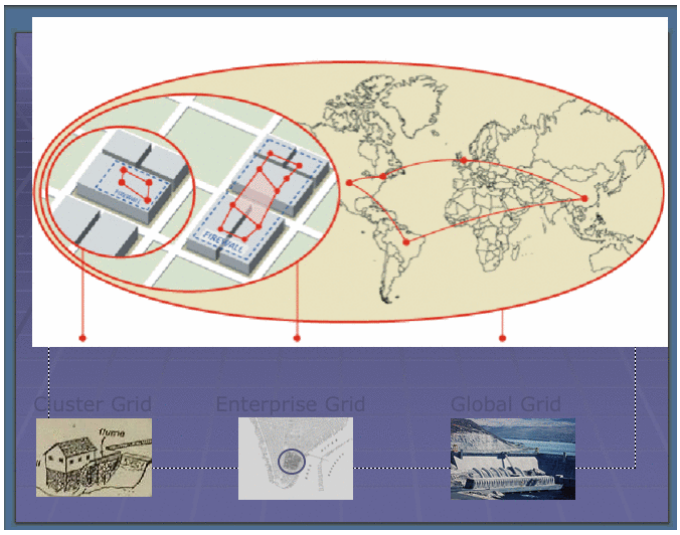


Figure 2-2: Logical deployment levels of Grid computing [7]

Global grids are set of enterprise, cluster grids and other distributed resources, and agreed upon protocols to facilitate resource sharing and global usage policies. *Enterprise grids* are used to facilitate multiple projects to share resources within the same enterprise.

A *cluster grid* is the simplest form of a grid. It provides a compute service to the department or group level. The software which enables this service might be a distributed resource management system, job management system, or job scheduling system. There are many benefits of using cluster grid like it enhance throughput for user tasks, and also maximize the use of computer resources.

Figure 2-3 shows the simplest grid topology. In which, we have 16 nodes in a grid topology. Each node is connected and interconnected with other node with eight four-port switches. This connectivity is related to star configuration, in which only two paths possible between

two nodes. This topology improves reliability and the simplicity of scaling the configuration to much larger computer calculation [37].

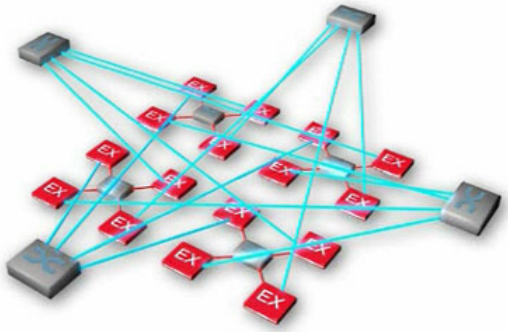


Figure 2-3: Grid topology [37]

Grid computing can be build in three kinds of topologies: [13]

- Intragrids,
- Extragrids
- Intergrids

The simplest topology is the *intragrid*, which exists within a single organization and which is consists of a basic set of grid services. The key characteristic of an intragrid is the single security provider. Bandwidth of intragrid on the private network is high and forever obtainable. It is easier to design and operate computational and data grids within an intragrid. It provides a comparatively static set of computing resources and the facility to simply share data among grid systems.

Extragrid enhances on the idea by taking together two or more intragrids based on a single organization. It involves more than one security provider, and the level of management complexity boost. The main characteristics of an extragrid are detached security, numerous organizations, and remote/WAN connectivity. Business would profit from an extragrid if there was a good idea to integrate with exterior trusted business associates.

An *intergrid* entails the dynamic integration of resources, applications, and services among patterns, customers, and any other official organization that will acquire access to the grid using the internet/WAN. An intergrid topology is mainly used by engineering firms, manufacturers, and by businesses. The major characteristics of an intergrid consist of dispersed security, and remote/WAN connectivity. A business might consider an intergrid essential if there is a necessity for peer-to-peer computing, or simplified end to end processes by the organizations that will employ the intergrid.

2.2 Cluster Computing

In last few years tremendous developments in fields of computer science like networking, software, buses, memory and microprocessors, making feasible for organizations to shrink or minimize their giant systems in micro computers, which are very effective in cost and have more strength, as compared to the old ones. Networking and cluster computing, though are not new, but in recent times those with other technological advancements are making high performing infrastructure for parallel and distributed applications [41].

A high-performance computing (HPC) cluster is a set of computing resources seen like a single resource with the end user. Computing cluster is used to run dynamic HPC applications across the resources and parallelized applications via message passing technology. It is also utilized to run any mixture of these applications [25].

High availability (HA) or fail-over clusters are designed to endow with uninterrupted availability of data or web services to the end-user community. The principle of these clusters is to make sure that a single instance of an application is only always running on one cluster member at a time except if and when that cluster member is no longer accessible, the application will fail-over to a different cluster member. High availability cluster implementations are best used in database application, file and print, mail web and application server. [6].

Open source cluster application resources (OSCAR) is an entirely integrated collection of software designed for building, maintaining, and using a Linux cluster for high performance computing (HPC). It allows users, despite of their understanding level with a "nix environment", to install a Beowulf type high performance computing cluster [24] [11]. In [24] author says that this software package (means OSCAR) has suit a well-accepted preference for building HPC clusters.

The current OSCAR cluster undergoes from several single point of failure and failure rates of some major components dominate the availability of the whole system. In order to recover the system availability, component redundancy is espoused in HA OSCAR cluster (The architecture of HA OSCAR cluster is shown in figure 2-4) to eradicate the single point of failure. Initial, a standby server is set up as duplication to capture the control of the cluster from the primary server when it is failed. A communication feature with heartbeat mechanism is useful as health detection among the two servers. Then, redundant network connection gives a substitute communication path in the case of switch, network card, or wire failure occurs. Last, a quorum voting mechanism is used for the multiple customers, and component redundancy will drastically enhance the system availability [9].

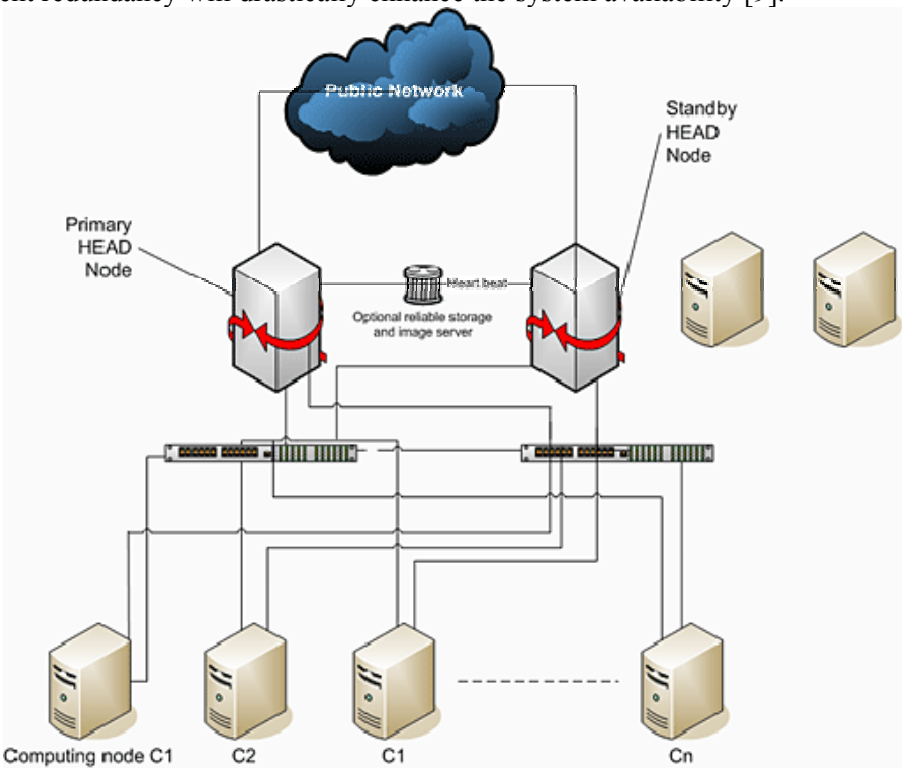


Figure 2-4: Architecture of HA OSCAR cluster [15]

Clusters key benefits are *scalability, availability, and performance*. On behalf of scalability, a cluster employs the collective processing power of compute nodes to run cluster-enabled applications, for example a parallel database server at a higher performance than a single machine can make available. Availability in the cluster is guaranteed as nodes within the cluster provide backup to each other in the occurrence of a failure. An extra benefit comes with the existence of a single system representation and relieve of manageability of the cluster. From the users viewpoint the users observe an application resource as the supplier of services and applications and the consumer does not recognize if this resource is a single server, a cluster, or even which node in the cluster is provided that services [6].

Fault-tolerance depicts a computer system or component designed so that, in the result that a component fails, a backup component can instantly take its position with no loss of service. Fault tolerance can be provided with software, or hardware, or combination of software and hardware [19]. In [22] author says, Distributed systems and clusters provide fault tolerance and high performance by using load balancing.

Load balancing cluster distributes incoming requirements for resources between multiple computers running the same programs. Each computer in the cluster is capable to handle requests for the same application. If a computer fails then requests are redistributed among the remaining free computer. This kind of division is typically seen in a web-hosting environment. Both, load-balancing and high availability cluster technologies can be shared to enhance the availability, and scalability of application [6].

Traditionally, parallel processing was performed by manifold processors in an especially designed parallel computer. These types of systems shared a single memory and bus interface within a single computer. By the beginning of high speed, low-latency switching technology, computers can be interconnected to make a parallel-processing cluster. These kinds of cluster enhance availability, performance, and scalability for applications, mainly computationally tasks. A parallel cluster is a system that uses a number of computers to simultaneously resolve a particular data mining or computational jobs. Its different to load balancing or high availability cluster, in which tasks are distributed to computer and computer process the entire request, and parallel environment is available here which divide the request into multiple sub tasks then these divided tasks are distributed to multiple computers in the cluster for processing. Parallel clusters are normally used for CPU-intensive analytical appliance, for example mathematical computation, scientific analysis, and financial data analysis [6].

3 PROBLEM DEFINITION/GOAL

The objective of our thesis is to apply new theoretical results of fault-tolerant load balancing to a modern distributed computing system. It concentrates on how recently discovered static recovery schemes can be applied to modern cluster and grid environments. The investigation is carried out by the help of simulation, including the comparison of Golomb and Trapezium recovery schemes with dynamic recovery schemes in clusters. Furthermore, we have tried to answer how static recovery schemes can replace a centralized scheduling approach.

Distributed computer systems can distribute task processing in the time of overloads. Load balancing involves the sharing of jobs right through a networked computer system, so increasing throughput without having to get supplementary or faster computer hardware. Load balancing policies may be either dynamic or static, Static load balancing policies are normally based on the information about the average activities of system; transfer decisions are independent of the actual current system condition. On the other hand, dynamic policies react to the actual current system state in making transfer verdict. This creates dynamic policies essentially more complex than static ones, and in actual fact optimal dynamic policies are recognized only for special systems [20]. This chapter focuses on dynamic load balancing strategies, taxonomy of dynamic load balancing, problems in dynamic load balancing strategies, static load balancing strategies and way in which we accomplish our goal.

3.1 Dynamic load balancing

After the initial assignment of job to each processor, dynamic load balancing is perform in four basic steps: [42]

- Monitoring processor performance
- Exchanging this information between processors
- Calculating new distributions and making the work movement decision
- The actual data movement directly among the slaves and the load balancing decisions are made by the load balancer.

Synchronization is triggered by the first processor that completes its part of the task, and this processor afterward sends an interrupt to all the other active slaves. This processor afterward sends their performance profiles to the load balancer. Load balancer calculates a new distribution when he receives all the profile information [30].

3.1.1 Dynamic load balancing strategies

Dynamic load balancing strategies have been shown to be the mainly critical fraction of a competent implementation of different algorithms on large distributed computing systems. Load imbalance can cause poor efficiency due to uneven processor utilization. Many dynamic loads balancing strategies have been projected in the last few years. It befalls harder for designers to compare and select the most appropriate strategy with this large number of algorithms. A load-balancing algorithm must pact with different unbalancing factors; unbalancing factors might be static, as in the scenario of processor heterogeneity, or dynamic. Examples of dynamic unbalancing factors are dynamic task creation, task migration, and variation of available computational resources. [30]

According to the first execution, the application fine-tunes itself for one of the load balancing strategies like: [30] [14]

- global-centralized (GCDLB)
- global-distributed (GDDLB)
- local-centralized (LCDLB)
- local-distributed (LDDLB)

The load balancer is placed on a central processor in Global Centralized (GCDLB), and the performance profiles are transmitting all through the system. The balancer sends individual instructions to every processor to control load balancing and computes the latest work load distribution.

The load balancer is imitation on all processors in Global Distributed (GDDLB), and the performance profiles are still transmitting all through the system. Individual instructions are not requisite to control load balancing. All load balancers have the requisite information desired to rebalance the system.

In Local Centralized (LCDLB) and Local Distributed (LDDLB), processors are divided into many different collection of size K. Performance profile information is exchanged only in the local group that reduced the amount of communication in the system as compared to global.

3.1.2 Taxonomy of dynamic load balancing

Dynamic load balancing is presented in the different ways from different points of views. In [30], Osman and Ammar introduced a good taxonomy of dynamic load balancing. A figure 3-1 represents the proposed taxonomy.

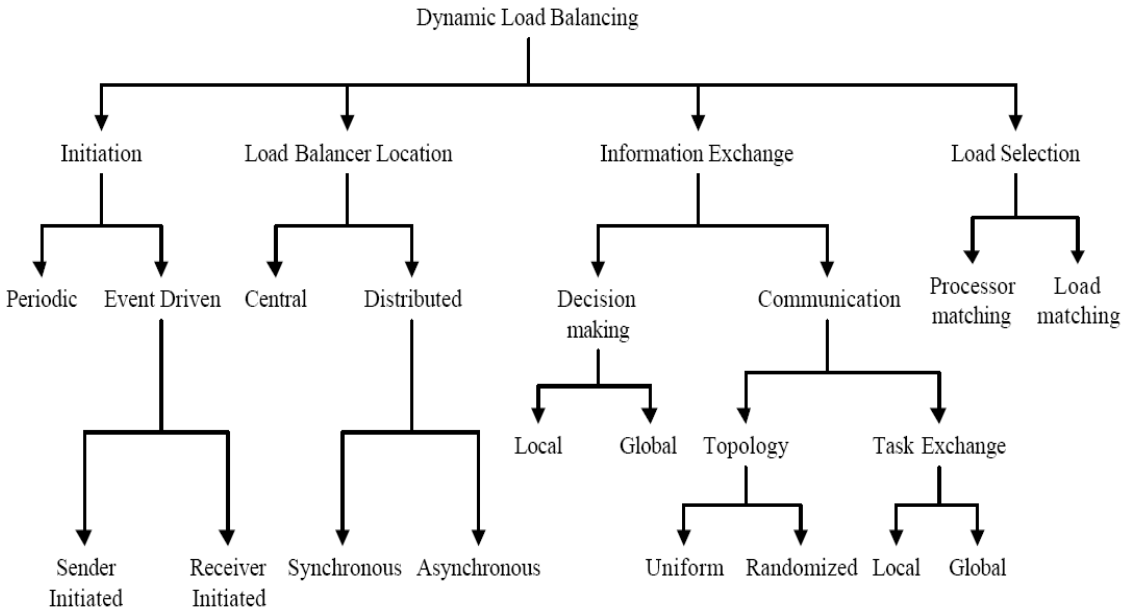


Figure 3-1: Taxonomy of Dynamic Load balancing [30]

Initiation strategy indicates the mechanism, which invokes the load balancing actions. Initiation strategy may be a periodic or event-driven. Periodic initiation is a timer based initiation in which load information is switched over each predestined time period. Event-

driven initiation is generally a load dependent strategy based on the surveillance of the local load where these load dependent strategies can be either sender-initiated or receiver-initiated**.** Sender-initiated means that over-loaded processors start activities even as the receiver initiated means that the under-loaded processors start the activities. Event-driven strategies are extra responsive to load imbalances. Periodic strategies are easier to implement and may result in extra overheads when the loads are balanced.

Load-balancer location strategy stipulates the location at which the algorithm itself is executed. It is said to be central if it is executed at a single processor, formative the essential load transfers and informing the concerned processors. If every processor takes part in the load balancing verdicts, the algorithm is categorized as distributed. Distributed algorithms are added categorized like synchronous and asynchronous. Although the employ of centralizing processor may guide to a bottleneck, it is significant to remember that distributed strategies necessitate load information to be proliferated to all the processors, foremost to higher communication costs.

Information exchange strategy stipulates the information and load flow through the system. The information employed by the dynamic load-balancing algorithm for decision-making can be local information on the processor or collect from the nearby vicinity. Global information exchange strategies tend to provide extra precise decisions, and local information exchange strategies may capitulate to less communication costs. The communication policy stipulates the connection topology of the processors in the system that decides the neighborhood of every processor. A randomized topology the processor erratically selects another processor to exchange information with, while uniform topology specifies a predetermined set of neighbors to communicate with. The communication policy also indicates the task/load exchange among different processors. Local strategies describe group of processors, and allow transfers to take place just among two processors within the similar group, while in Global strategies, task/load transfers may take place among any two processors.

Load selection strategy stipulates the processors involved in the load exchange (processor matching). It specifies the suitable load items (load matching) to be exchanged. Local averaging represents one of the common techniques. When overloaded processor own load drops to a precise threshold then it transfer packets to its neighbors.

Although above four major sections needs research when we talk about static recovery schemes. This paper contributes results according to above taxonomy in Load-balancer location strategy. Simulation is also done as distributed system in asynchronous mode.

3.1.3 Advantages and disadvantages with Dynamic Load balancing

Data-intensive and complex database queries consent parallel processing strategies to attain adequately less response times. Parallel database processing is mostly based on shared nothing architectures necessitate a physical partitioning and allocation of the database between multiple processing nodes. The main characteristic of shared nothing cluster is that the database is partitioned between all computers so that every processing element (PE) has one partition [26].

Dynamic load balancing has problem in Shared Nothing database systems. Load balancing is a critical requirement for efficient utilization of super servers, and it is used to support effective intra-query parallelism in multi user environment. The main control decisions to describe dynamically comprise determining the amount of intra-query parallelism and used to choose the processors for executing sub queries. [35]

Numerous companies provide recovery schemes for both dynamic and user defined recovery schemes in case of crash, and random worst-case behavior and comparatively long switch-over delay of specific dynamic schemes is unappealing in various (real-time) applications. Besides, system cannot use dynamic load balancing schemes when the primary destination for their input is not responding. The large shared nothing clusters should know where the task will move if the current node crashes. The reason for this is that all executed data is copied to the computer on which the job will restart in case of crash, and for large cluster system it is impossible to copy all data to each node in the cluster. This indicates that we should use static schemes for large shared nothing clusters. So finding optimal recovery schemes is thus a primary dilemma in distributed and cluster systems [22].

3.2 Static Load balancing

Static load balancing determines the individual processor, loads prior to runtime, and it provides best to applications that run on dedicated systems with predefined feature. Tasks are typically partitioned accordingly at compile time. Tasks are assigned due to influenced by the problem size, and relative performance of individual processors, etc. However, once the tasks are assigned, and unable to change at runtime [14]. We focus on two static load balancing schemes like:

- Golomb recovery scheme
- Trapezium recovery scheme

We discuss about Golomb and Trapezium recovery schemes in detail in chapter 5. In our thesis, we compare and analyze static recovery scheme like Golomb and Trapezium.

3.2.1 Advantages and disadvantages with static load balancing

The purpose of recovery schemes in a distributed system, either static or dynamic, is to improve performance by redistributing the load among active nodes. There are merits and demerits of static load balancing in distributed system, and it depends on the situations, recovery schemes, network speed, load balancing policies and etc.

Static load balancing are easy to implement and does not require complex decision to make. In static load balancing, transfer decision is clearly mention and no overhead is required in searching the active node from whole given system. Extra processing is not required to keep consideration which node is active and which has failed. This processing time can be utilized in other tasks. Load can be evenly distributed in worst case with good recovery scheme which increases the overall performance.

Static load balancing are commonly based on the information about the average behavior of system. There are different scenarios when static load balancing can be average, best or worst. In case of failure of node; reassignment / transfer decisions are independent and can become the worst case in some situation. If the capability or capacity is low in given static load balancing; and job need higher capacity machine then job will never execute. In static load balancing all information about workstation / nodes must be available prior to execution that can be a pitfall in a distributed system.

3.3 Simulation Toolkit

Testing dynamic and static new recovery schemes (Golomb and Trapezium) by simulations and their result analysis (best, equal, worst) are related to empirical investigations. For simulation we have analyze many toolkits like Gridsim, SimGrid, BeoSim, Sim BOINC. There is comparison between Gridsim and SimGrid: [34]

- According to our analysis GridSim and SimGrid are same but in some sense, GridSim is a higher-level simulator compared to SimGrid. It is basically designed to investigate interactions and interferences between scheduling decisions taken by distributed brokers. [34]

- GridSim was proposed and designed after SimGrid. Its motivations are quite similar. One main difference concerns its focus on Grid economy, where the scheduling involves the notions of producers (resources owners), consumers (end-users) and brokers discovering and allocating resources to users. [34]

- GridSim is mainly used to study cost-time optimization algorithms for scheduling task farming applications on heterogeneous Grids, considering economy based distributed resource management, dealing with deadline and budget constraints. [34]

- Although based on an already established simulation platform, GridSim is not as methodological in simulating realistic network topologies, link congestion, resource contention, and parallel applications as SimGrid [36].

Table 3-1: Comparison between different toolkits [32]

Functionalities	GridSim	MicroGrid	SimGrid	OptorSim
Routing table entry	Automatic	Automatic	Manual	Manual
Type of transport protocol	A datagram oriented protocol	TCP and UDP	TCP	Not supported
Data packetization	Supported	Supported	Not supported	Not supported
Runtime Network status	Supported	Supported	Not Supported	Not Supported
Network QoS	Supported	Not Supported	Not Supported	Not Supported

So from the above table 3-1, we can predict that Gridsim have more functionality. We have selected Gridsim for simulation because it provides better functionalities [32] as compare to other toolkits.

4 METHODOLOGY

Research is an organized way to find the elucidation of given problem or we can express like that to find the solution/answer of question in structured way. When anybody investigate particular problem, they always understands the background, questions and the approach they are starting, which are useful for finding that objective.

In order to find out the solution in attempt to answer the specific questions, qualitative & quantitative research methodology will be used.

In qualitative part a detailed and comprehensive literature study will be carried out to gather material related to grid computing, cluster systems, schedulers, fault tolerant system, static recovery schemes e.g. Golomb and Trapezium.

In quantitative analysis we have done simulation using Gridsim tool for analyzing how Golomb and trapezium recovery schemes work in an optimal scenario as compare to dynamic recovery schemes.

5 THEORETICAL WORK

5.1 Distributed Computing

The last two decades gives a revolution in the field of computing, and computing world shifts from central mainframe-based computing to network- based computing. Nowadays, servers are being used to resolve the computationally intensive problems in science, and they are also used to attaining fast levels of CPU performance, memory capacity and I/O bandwidth. Distributed computing is built from a collection of heterogeneous computers, workstations, or server and it behaves like a single computing system. In distributed computing system, users can consistently access and name (local or remote) resources and execute their jobs anywhere in the system without knowledge of which process are running on this computer. We will get many benefits by using distributed computing due to advancement in processing, networking technology and software tools: [27]

- By using distributed system performance is increased because existence of many computers in a distributed system gives permission to the applications to be processed in parallel and therefore enhance application and system performance. The performance of file system is enhanced due to replication of its function over several computers. In addition, file replication is also providing many benefits like reduced network contention and queuing delays.
- Distributed system provides facility of sharing resources. In distributed systems, users are able to share expensive hardware and software resources such as printer servers, compute servers, database servers etc.
- Distributed system also increase extendibility, due to limitation in file system capacity and computing power can be overcome by adding more computers and file servers to the system incrementally.
- System reliability, availability, and fault tolerance are increased due to Distributed system. Distributed System is able to tolerate the failure of one computer by give out the jobs to other free computer in the system.
- Distributed system is cost-effectiveness and the performance of computers has been double up in every two year but the cost has decreased by half in each year during the last decade. In addition, the new high-speed network technology will compose the development of distributed systems attractive in terms of the price and performance ratio as comparison to the parallel computers.

5.1.1 Distributed system design framework

Distributed system design framework (DSDF) is used to implement the major components of any distributed computing system. It describes the architectural issues, candidate technology and services. Design process of distributed computing entails three main activities: [16]

- Establishing the communication system that helps to interchange information between distributed system resources and objects.
- Make the organization and system services that establish several computers to act as a system rather than group of computer.
- Creating the distributed computed programming techniques to establish parallel and distributed applications.

Distributed system design framework can be illustrated in terms of three layers and these also based of design process: [27]

i. Network protocol and Interface (NPI) layer
ii. System architecture and services (SAS) layer
iii. Distributed computing paradigms (DCP) layer

<table>
<tr><th colspan="5">Distributed Computing Paradigm</th></tr>
<tr><td colspan="2">Computer Models</td><td colspan="3">Communication Model</td></tr>
<tr><td>Functional Parallel</td><td>Data Parallel</td><td colspan="2">Message Passing</td><td>Shared Memory</td></tr>
<tr><th colspan="5">System architecture and services (SAS)</th></tr>
<tr><td colspan="2">Architecture Model</td><td colspan="3">System-level services</td></tr>
<tr><th colspan="5">Network Protocols and Interface (NPI)</th></tr>
<tr><td>Communication protocols</td><td colspan="2">Network Type</td><td colspan="2">Network interfaces</td></tr>
</table>

Figure 5-1: Distributed system design framework [27]

Network protocol and Interface (NPI) layer illustrates the key components of communication system and these communication system will be used for shifting control and information between the distributed system resources. NPI layer has three sub layers, which are: communication protocols, network type and network interfaces.

System architecture and services (SAS) layer characterize the system managers and designers view of the system. SAS layer have to support by the distributed system because it provides a single image of computing system. SAS layer describes the structure, architecture load sharing and load balancing.

Distributed computing paradigms (DCP) layer, main focus on the programming paradigms that can be utilized to develop distributed applications, and it corresponds to the user perception of the distributed system. DCP layer is based on the computation and communication models. Distributed and parallel computations can be depicts in two paradigms such as data parallel and functional parallel paradigm. In data parallel paradigm, each computer works on different data streams but every computer runs the same program, same program multiple data stream (SPMD). In functional parallel paradigm, computations are split into discrete functions which are afterward allocated to unlike computers. Inter-task communications is used to characterize parallel and distributed computing, and it has two main models: distributed shared memory and message passing models. In distributed shared memory, tasks correspond by reading/writing to a global shared address space while in message passing; tasks correspond through messages.

5.2 Static Recovery Schemes

In the thesis, we consider a cluster where every computer holds one process and, work is equally distributed among these computers (processors). Each process has its own recovery list with the computers on which it should be resumed if its own processor (computer) will be damaged. Below fig 1 shows a system for four computers. We assume that processors are moved back to computer when its working properly, this can be configured automatically but mostly done by user. The proper working system is shown on left side of figure. One process is on every computer and every process hold a recovery list. The right side of Figure 5-2 shows that when computer 0 damage, it should be restarted on computer 1due to recovery list (1, 2, 3) that is connected with computer 0. In this scenario, if computer 1 also breaks down then process 0 will be shift to computer 2 because it's the second computer in the

recovery list. Now process 1 is shifted to computer 3 because first computer of recovery list is already broken. Therefore, if computer 0 and 1 are down then there are two processors on computer 2 and two process on computer 3. This is an efficient result as the load is evenly distributed among two computers. Now, if computers 0 and 2 break down then three processors are shifted to computer 1 and one process on computer 3. In this scenario, load on computer 1 is three times the original load, and this is most unfavorable combination [22].

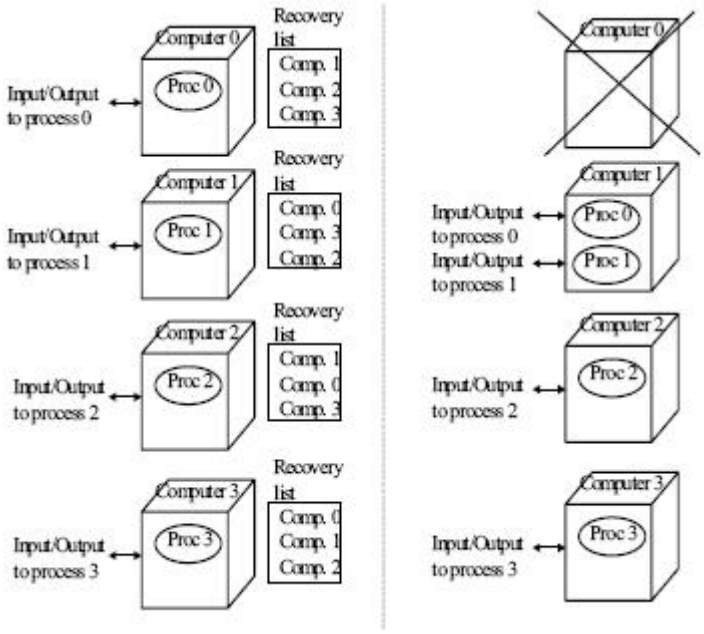


Figure 5-2: Application executed on cluster with four computers [22]

Static recovery schemes are also applicable in the external systems that are feeding data in a cluster e.g. telecommunication switching center. In a case of computer break down, the switching center transfer data to some other computer by using recovery list that is associated with each computer [22].

The large shared nothing clusters should know where the task will move if the current node crashes. The reason for this is that all executed data is copied to the computer on which the job will restart in case of crash, and for big cluster it is impossible to copy all data to each node in the cluster. This indicates that we should use static schemes for large shared nothing clusters. So finding optimal recovery schemes is thus a primary dilemma in distributed and cluster systems [22]. Now, we analyze the working of Golomb and Trapezium recovery schemes.

5.2.1 Golomb Recovery scheme

“In mathematics, the term "Golomb Ruler" refers to a set/sequence of non-negative integers such that no two distinct pairs of numbers from the set/sequence have the same difference” [33]. In Golomb ruler, no two pairs of integers (also called marks in the sequence) measures the same distance. Optimal Golomb Ruler (OGR) is the shortest Golomb Ruler. However, it is difficult to find an Optimal Golomb ruler when number of marks increases [33].

Optimal Golomb ruler has a lot of applications like sensor placements for X-ray crystallography and radio astronomy [33]. Golomb ruler provides a way to place marks along a line. Golomb ruler measure a unique linear distance in each pair of marks [22].

Here is a Golomb ruler with five marks: [33]

```
| | |  | |
0 1 4  9 11
```

The numbers below the marks shows the distance from left to right. The length of above Golomb ruler is 11. The distances in this example (Table 5-1) can be presented as a set of pairs as follows: [33]

Table 5-1: Optimal Golomb ruler's distances represented by pair of marks [33]

Mark1	Mark 2	Distance
0	1	1
0	4	4
0	9	9
0	11	11
1	4	3
1	9	8
1	11	10
4	9	5
4	11	7
9	11	2

Looking at the last column (distance) we see that all numbers are unique. However, number 6 is missing. [33]. Another way of representing Golomb ruler is by using the differences/distances between the marks that shown in below figure 5-3:

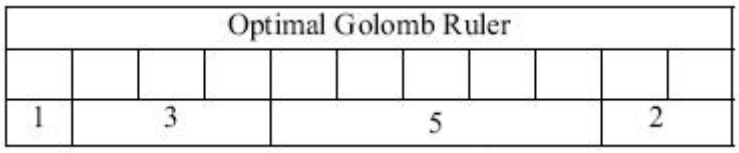


Figure 5-3: Optimal Golomb ruler (OGR) represented by differences [22]

By comparing these three representation we can see Golomb ruler as a triangle (see below Figure 5-4), in which the first row corresponds to the marks (see Figure 5-2), the second row to the distances (see Table 5-1) and the following rows corresponds to other differences (see Figure 5-3).

```
0  1  4  9  11
   1  3  5  2
      4  8  7
         9  10
            11
```

Figure 5-4: Triangle presentation of OGR [22]

The complexity of finding optimal Golomb ruler (OGR) is very big, so there is only 24 Optimal Golomb rulers are available till now i.e. shown in Appendix 1.

Golomb recovery list for a computer 0 is obtained from the Golomb ruler in the following way: [22]

- First part of recovery list is acquiring through Golomb rulers

- Second part consists of the remaining number of computers up to *n - 1*, where n is the number of computers in the cluster

The recovery lists for the following computers are obtain by adding the number of computer to the number from the recovery list for computer 0 mod n. For example, for n = 12 we will use OGR <0,1,4,9,11>. Then for each computer we build the following recovery lists: [22]

Computer 0: 1,4,9,11,2,3,5,6,7,8,10
Computer 1: 2,5,10,0,3,4,6,7,8,9,11
Computer 2: 3,6,11,1,4,5,7,8,9,10,1
Computer 3: 4,7,0,2,5,6,8,9,10,11,2
Computer 4: 5,8,1,3,6,7,9,10,11,0,3
Computer 5: 6,9,2,4,7,8,10,11,0,1,4
Computer 6: 7,10,3,5,8,9,11,0,1,2,5
……..
…….
Etc.

5.2.2 Trapezium Recovery scheme

We have taken idea of Trapezium recovery scheme from [23].

The Trapezium recovery scheme belongs to the second static recovery scheme described in this thesis. They provide better optimality, performance and are simple to calculate as compare to Golomb recovery scheme. In Trapezium recovery scheme, first part is based on Golomb ruler and second part is made from a new way.

Let S be a sequence of positive integers that shows the distances $S = (s_1, s_2, \ldots . s_{n-1})$. In a case of Golomb ruler, this is the sequence of the differences, e.g. $S = (1,3,5,2)$.

Let C_i be a crash rote for computer i in the cluster defined in the following way:

crash route C_1: s_1,
crash route C_2: (s_1+s_2), s_2,
crash route C_3: $(s_1+s_2+s_3)$, (s_2+s_3), s_3,

Let now the j:th entre in the crash route C_i be denoted by $C_{i,j}$, e.g. i.e. $C_{3,2} = (s_2+s_3)$.

The Trapezium recovery list is defined in the following algorithm [23]:

Given a number (n) – find the longest sequence $S = < s_1, s_2, \ldots . s_m >$ *of positive integers such that:*

a) *the sum of the elements in the sequence is less than equal n*
b) *l is an integer number such as:* $l > (\sqrt{8k+9} - 3)/2$, *where k is the number of crashed nodes*
c) *the first l crash routes are disjoint*
d) *the following* $l(l+1)/2 - l$ *crash routes have at least l different values compared to the previous crash routes*

Figure 5-5 shows a triangle representation of the sequence $S = < 1, 3, 2, 5, 2, 5 >$ for $n = 19$ with its crash routes. The bold numbers are all different. The first three crash routes are disjoint i.e. $C_1 = \{1\}$; $C_2 = \{4, 3\}$; $C_3 = \{6, 5, 2\}$ (this is an OGR). In the following crash routes there are minimum 3 unique values (i.e. $l = 3$) and, its easily calculate, the number of crashed nodes is less than 9. Thus, the sequence should be optimal for up to 8 crashed nodes.

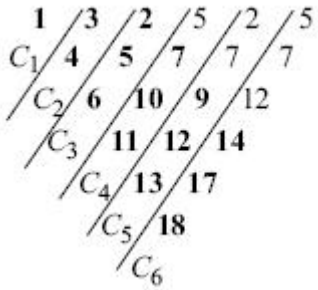


Figure 5-5: Triangle representation of the sequence S = < 1, 3, 2, 5, 2, 5> [23]

From Trapezium algorithm we build a recovery scheme in the same manner as for the Golomb recovery scheme.
First we construct the recovery list R_0 for the computer number 0 as a sequence of distances:

$$R_0 = (0, s_1, s_1 + s_2, s_1 + s_2 + s_3, \ldots, s_1 + \ldots + s_{n-1})$$

The recovery lists for the other computers are made as follow:

$$R_i = \{(r_0 + i) \bmod n, \ldots, (r_n + i) \bmod n \} \text{ for } 0 \leq i < n,$$
where $r_i = s_1 + \ldots + s_i$ is the i:th position in the vector R_0.

Here we show an example for a sequence $S = <1,2,2>$. In this sequence number of computers in a cluster is 6. Recovery lists for all computers are presented below:

$R_0 = 1, 3, 5$
$R_1 = 2, 4, 0$
$R_2 = 3, 5, 1$
$R_3 = 4, 0, 2$
$R_4 = 5, 1, 3$
$R_5 = 0, 2, 4$

6 EMPIRICAL STUDY/CASE

In this chapter we discuss various simulation tools like the reason of choosing Gridsim, Gridsim characteristic, Gridsim entities, Gridsim architecture and the results of our simulations.

We demonstrate the effectiveness of associated scheduling algorithm with static recovery schemes like Golomb and Trapezium and their performance evaluated under different cases, varying number of users and resources with different requirements. It is almost nearly impossible to perform scheduled performance in a repeatable way for different scenarios in a grid environment. There are many reasons of this, such as the availability of resources and their load which continuously varies with time. It is impossible for an individual who is in one domain to control the activities of the users who are in different administrative domains [3].

6.1 Simulation Tools

Simulation of Grid / cluster is a vast area of research, and in reality it is not erroneous to say that Grid infrastructures are becoming the most intricate distributed systems in the world. Therefore it is not astonishing that there exist several different simulation tools. But a big issue to any simulation is the question of validity: Does a simulation articulate that what would take place in real world? So that we can see that there is no ideal simulation tool for all kinds of simulations of any broad scientific area [12]. Hence there are various simulation tools, which each are focused on special purpose in special category of grid / clusters. An outline of some toolkits is given below: [29]

- Bricks
- SimGrid
- Gridsim
- Optorsim
- ChicagoSim
- Chicagosim and Parsec
- Edgsim and Ptolemy II
- Huskysim

Bricks [12] is a first Grid resource scheduling simulator and it is written in Java. It is a performance evaluation system for scheduling algorithms [38]. In Bricks [12], simulation has been validated by comparison of measurements on a simulated Grid and real monitoring. It defines different network topologies, and processing schemes of servers and networks [38, 12].

We are comparing different tools, so in one sense its empirical study, empirical means Research that uses data drawn from observation or experience.

OptorSim is used to designed test dynamic replication strategies. It is used in optimizing location of data within a Grid [29].

ChicagoSim is also called Chicsim, is built on top of parsec. Parsec is a chased simulation language developed for parallel environment. ChicagoSim allows plugging in of arbitrary algorithms into a holistic Grid simulation [12].

EDGSim is a simulation toolkit which is written in Java-based Ptolemy II heterogeneous concurrent design and modeling software. Ptolemy is a framework that simulates system that consists of heterogeneous components and sub-components [12]. According to [5], it is designed to simulate the European DataGrid.

HuskySim is used to simulate both static and dynamic job execution assignment to different processors then hybrid approach is used to combine static and dynamic techniques. It is developed in Java [12].

SimGrid is used for the simulation of heterogeneous and volatile desktop grids and volunteer computing systems. Its goal is to make possible research in the field of distributed and parallel application, scheduling on distributed computing platforms that are ranging from simple network of workstations to Computational Grids. SimGrid basically simulate distributed application in heterogeneous distributed environment [12].

6.2 GridSim Description

GridSim is a simulation toolkit that allows modeling of entities in parallel and distributed computing, resources and applications for evaluation and design of scheduling algorithms. GridSim provide facility for creating heterogeneous resources that can be used to solve data intensive applications. GridSim simulates millions of resource and thousands of users with different requirements, and studies scalability and efficiency of schedulers.

With the help of GridSim we can get simulation of single and multiprocessor computers. Heterogeneous resources can be accessed virtually as time or space shared systems. It ropes entities that simulate individual computers and networks used for communication between resources. GridSim creates various multi-threaded entities during simulation, and all of them execute in parallel, in its own thread [4].

6.2.1 GridSim entities

GridSim based simulations contain entities for the users, brokers, grid resources (computing resources), information service, statistics, and network based I/O as shown in Figure 6-1. From the below figure we can get the basic idea how GridSim entities work [4].

Each occurrence of User entity represents a Grid user. Each user may vary from other users on the behalf of given characteristics: types of job created, Scheduling optimization strategy and activity rate [4].

Each broker instance is coupled with user entity. Each time when user submits a job, its first submitted to its brokers and then broker schedules according to the predefine user scheduling policy.

Every occurrence of the Resource entity represents a grid resource. Each resource may have diverse number of resources with respect to Number of processors, Cost of processing, Speed of processing, Internal process scheduling policy e.g., time shared or space shared, Local load factor and Time zone.

Grid Information Service (GIS) is responsible for resource registration services and organize list of resources available in the Grid [4].

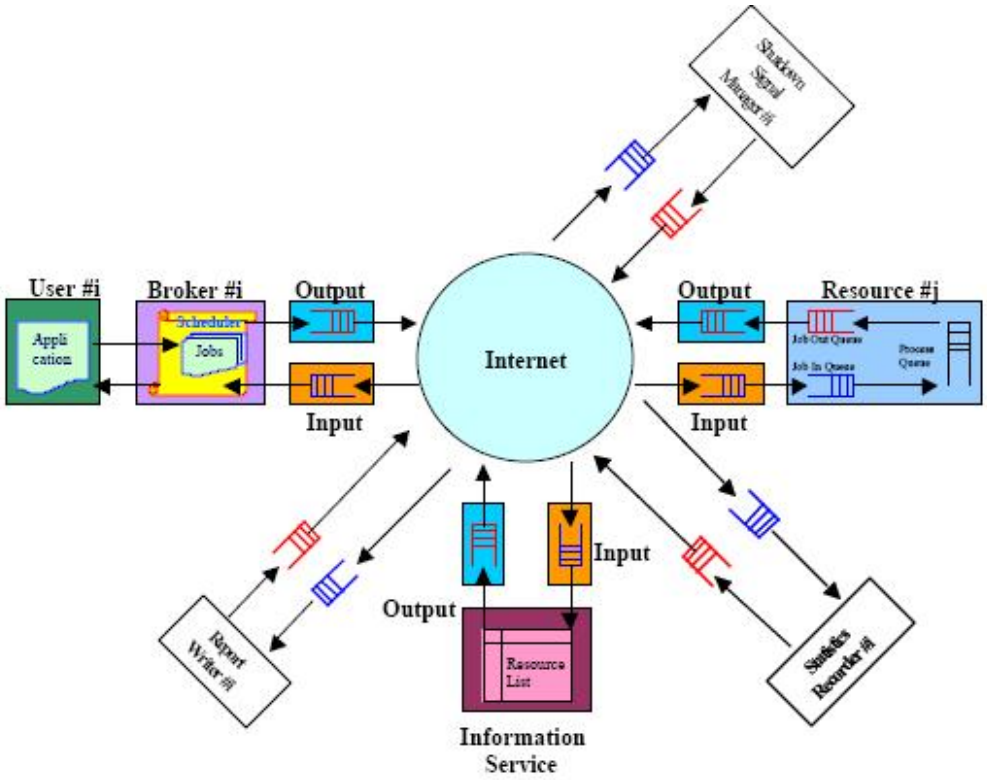


Figure 6-1: Flow diagram in GridSim based simulations [4]

6.3 Major Simulation Results

In this section we present simulation results. How we have got results using different numbers of computers and how we have measured performance.

6.3.1 Simulation Setup

For the analysis of our results we have set many constants and some variables as well. To predict the exact analysis of Golomb and Trapezium with dynamic schedulers, we have different number of computers.

As we have discussed GridSim entities in section 6.2.1, now we can get the clear idea of how these GridSim entities work in reality.

Consequently the constants that we are using in our simulations are here:

- The cost used in migration of process is always 5 £, it is used to check the internal total cost of each resource in a grid.
- Creation of Gridlet / task and their assignments time is always set to be zero.

Furthermore:

- Allocation of resource time varies individually and it depends on the turnaround time of each individual task / Gridlet in the system.
- Migration time is calculated when we subtract the submission time from crash time.
- The turnaround time is the total time that each task / Gridlet takes for the completion of job. We have calculated turnaround time for each

individual Gridlet / task and final turn around time is the time, when all 7 computers finish their job / task.

6.3.2 Scenario with 7 computers, 01 Processing Element (PE) in Static recovery schemes

In the tables below (table 6-1) we present the examples of the simulations with all variables that we have accomplished having 7 computers with one processing element (processors) in it, using Golomb, Trapezium and FIFO dynamic scheduler.

In our simulation, the major considerations of analysis are Execution time, Crash time, turnaround time. Beside this each description of table is given below.

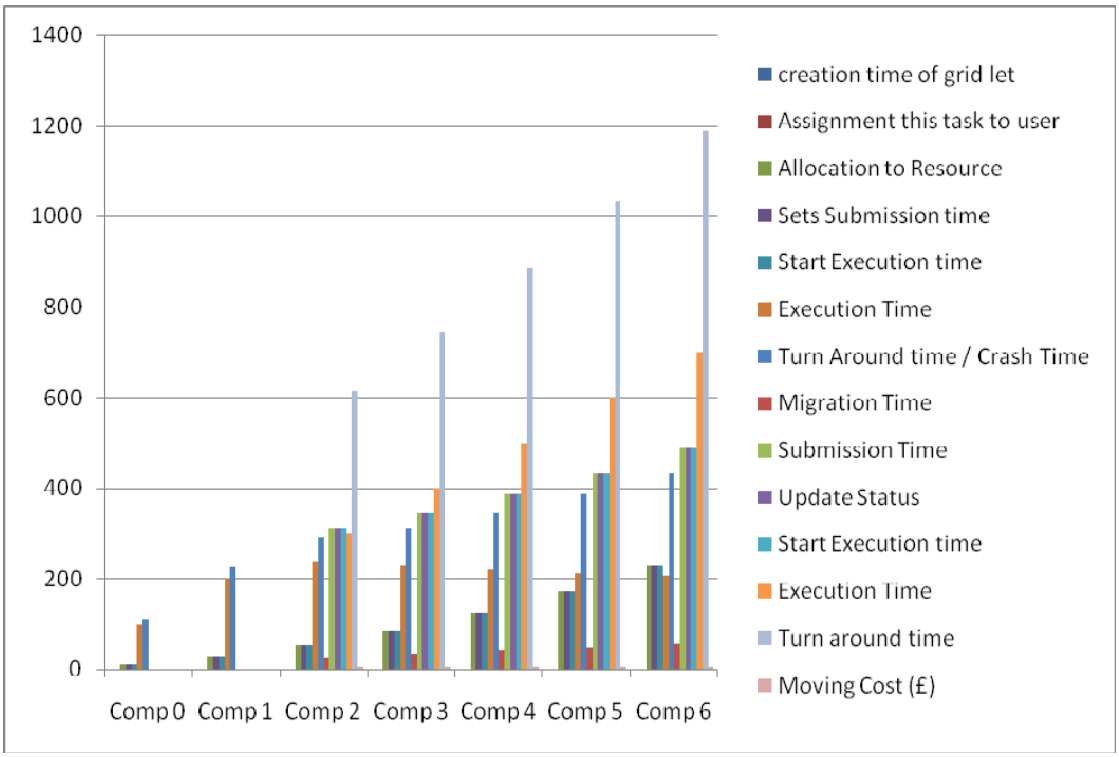
Before simulation starts, gird resources, gridlet and processing element have been created. Creation of gridlet always starts from 0^{th} second. Line number two indicates assignment to user; it's always set to zero seconds in simulation. Allocation of resource is the time when each task gets resource for processing. Set submission time is time when the task or job submits in queue; this indicates that it has submitted in queue for processing. Start execution time is also same as allocation to resource time. Execution time is processing time that each task gets individually from processor or processing element. Turn around time is the total time of each task / gridlet. Crash time is the time when any machine crash or fail during processing. Migration time is the time when the task shift from one machine to another. Moving cost is set to 5 £ and it is use for internal measurement of simulation.

The yellowish row in tables represents the actual execution time of each computer spent for processing. Red color indicates the failure or crash of each computer at particular time, and if any computer crashes, it doesn't mean it has ended; it needs further resources and processing element for execution. The green color represents the turn around time, which is basically the time we are interested in. Total turn around time is the time which is being compared between static and dynamic schemes.

Table 6-1: Crash and turnaround time of 7 computers (Static Scheme)

Description	Comp 0	Comp 1	Comp 2	Comp 3	Comp 4	Comp 5	Comp 6
creation time of grid let	0	0	0	0	0	0	0
Assignment this task to user	0	0	0	0	0	0	0
Allocation to Resource	11	27.064	51.128	83.192	123.256	171.32	227.384
Sets Submission time	11	27.064	51.128	83.192	123.256	171.32	227.384
Start Execution time	11	27.064	51.128	83.192	123.256	171.32	227.384
Execution Time	100	200	237.12	229.224	221.366	213.488	205.56
Turn Around time / Crash Time	111	227.064	288.24	312.41	344.59	384.76	432.94
Migration Time			24	32.006	40.002	48.008	56.004
Submission Time			312.24	344.416	384.592	432.768	488.944
Update Status			312.24	344.416	384.592	432.768	488.944
Start Execution time			312.24	344.416	384.592	432.768	488.944
Execution Time			301.08	400.08	500.08	600.08	700.08
Turn around time			613.328	744.504	884.68	1032.856	1188.944
Moving Cost (£)			5	5	5	5	5

Hence we can depict from Table 6-1 that the turnaround time of last task \ computer is 1188.944 seconds, when we have 7 computers and 5 computer have been crashed by using Golomb and Trapezium recovery schemes. The results are obtained using static recovery scheme and when any computer like 2nd will crash then the resources will shift according to static recovery scheme i.e. 3rd computer. There are no overheads at runtime, no need to keep knowledge of all active or fail computers and etc information for scheduler. When any computer will fail it will be handle according to predefine scheme. The graphical representation of these data can also be seen in graph 6-1.



Graph 6-1: Crash and turnaround time of 7 computers (Static Scheme)

From graph 6-1 we can see the turnaround time and other measurement of our simulation.

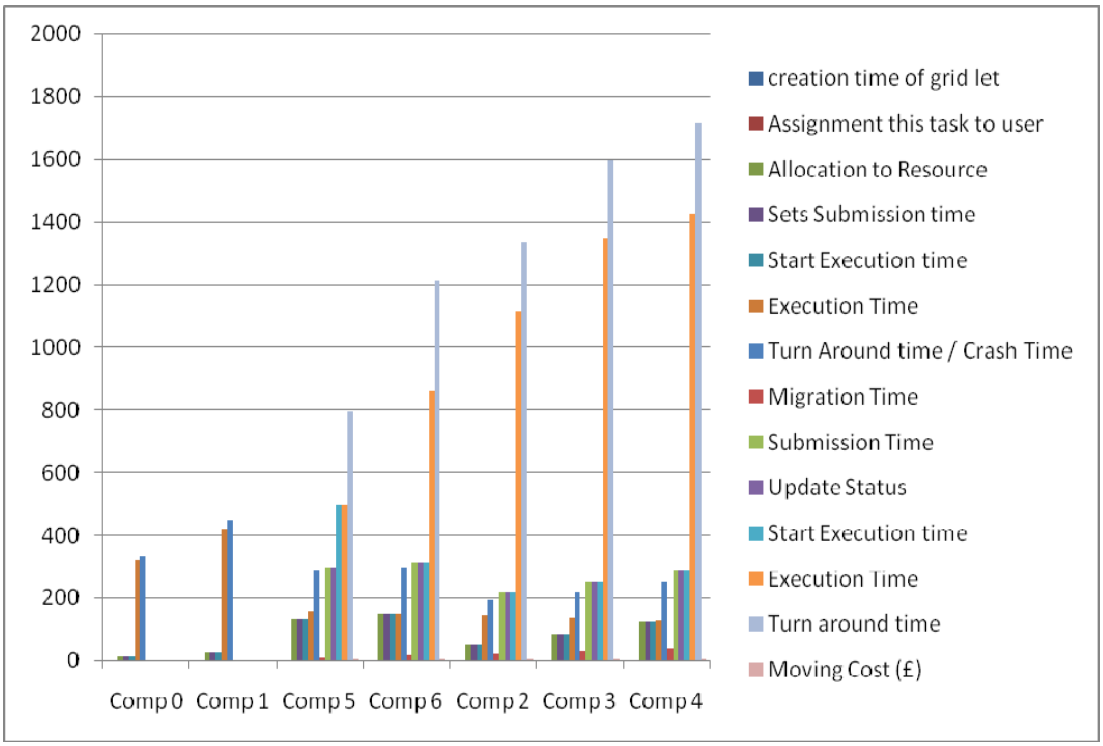
6.3.3 Scenario with 7 computers, 01 Processing Element (PE) in Dynamic Scheme

Table 6-2 presents a similar scenario as Table 6-1, i.e. the same number of users, resources, processing element, but with dynamic scheduler policy. Here the total turnaround time is 1716.56 seconds, which is longer as compare to Golomb and Trapezium.

Table 6-2: Crash and Turnaround time of 7 computers (Dynamic Scheduler)

Description	Comp 0	Comp 1	Comp 5	Comp 6	Comp 2	Comp 3	Comp 4
creation time of grid let	0	0	0	0	0	0	0
Assignment this task to user	0	0	0	0	0	0	0
Allocation to Resource	11	27.064	131.32	147.384	51.128	83.192	123.256
Sets Submission time	11	27.064	131.32	147.384	51.128	83.192	123.256
Start Execution time	11	27.064	131.32	147.384	51.128	83.192	123.256
Execution Time	320.7	420.7	157.56	149.88	141.43	133.54	125.65
Turn Around time / Crash Time	331.709	447.773	289.08	297.64	192.56	216.76	248.912
Migration Time			8.008	15.624	24	31.976	40
Submission Time			297.088	313.264	216.56	248.736	288.912
Update Status			297.088	313.264	216.56	248.736	288.912
Start Execution time			495.95	313.264	216.56	248.736	288.912
Execution Time			495.95	859.88	1115.56	1347.47	1427.68
Turn around time			793.044	1209.22	1332.12	1596.472	1716.56
Moving Cost (£)			5	5	5	5	5

Here we have got our result from dynamic scheduler. In this case a central scheduler is taking care of resources and computers. If any computer will crash then the central scheduler will monitor it and move the job to another active computer, but if that particular computer is already crash then scheduler will transfer to another computer. So computer 0^{th} and 1^{st} has completed successfully but as computer 5^{th} crashed the resources has been shifted to computer 6 etc. Hence the total turnaround time is longer as scheduler has other complex task to perform. The graphical representation of these data can also be seen in graph below.



Graph 6.2: Crash and turnaround time of 7 computers (Dynamic Scheduler)

From above graph we can see the turnaround time of all computers with other measurements as well.

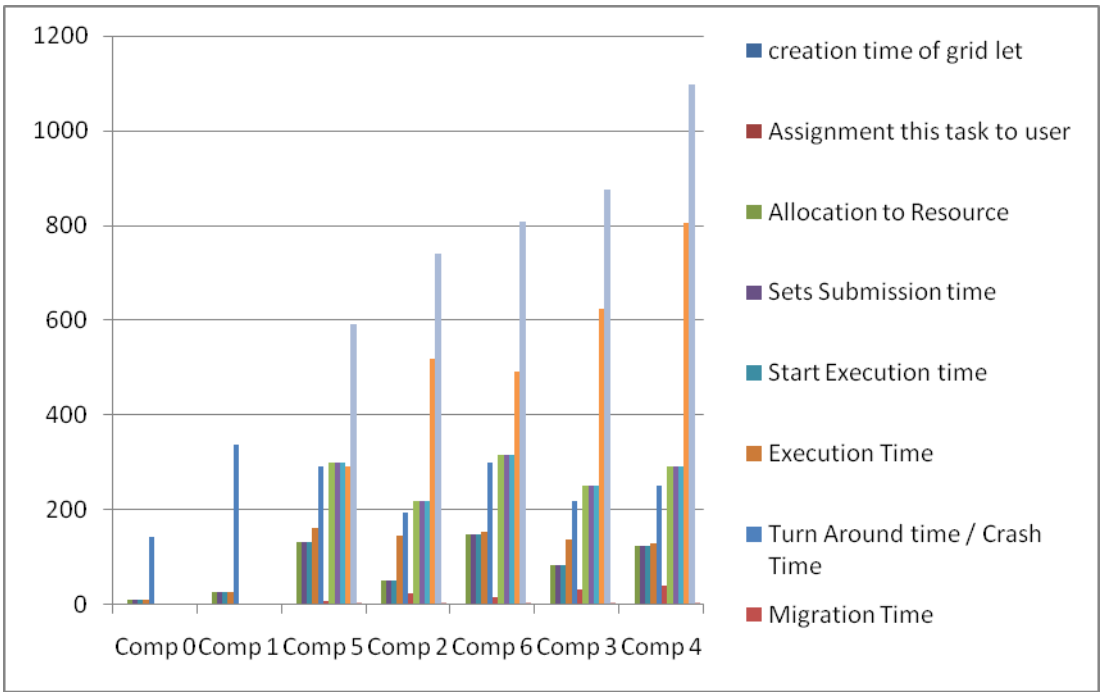
6.3.4 Scenario with 7 computers, 02 Processing Element (PE) in Dynamic Scheme

If we increase the number of processing element or machines in each computer then dynamic scheduler work better then Golomb and Trapezium, so the turnaround time of 7 task / Gridlet is 1096.50. In theory if we have more than one processing element or (*n*) machines with static recovery schemes then we should use two or *n* recovery schemes for each processing element or machines. However, this is not the drawback of Golomb or Trapezium recovery schemes.

Table 6-3: Crash and turnaround time of 7 computers (Dynamic Scheduler with different PE

Description	Comp 0	Comp 1	Comp 5	Comp 2	Comp 6	Comp 3	Comp 4
creation time of grid let	0	0	0	0	0	0	0
Assignment this task to user	0	0	0	0	0	0	0
Allocation to Resource	11	27.064	131.32	51.128	147.384	83.192	123.256
Sets Submission time	11	27.064	131.32	51.128	147.384	83.192	123.256
Start Execution time	11	27.064	131.32	51.128	147.384	83.192	123.256
Execution Time	11	27.064	160.936	144.6	153.048	136.712	128.824
Turn Around time / Crash Time	142.552	337.469	292.256	195.728	300.432	219.904	252.08
Migration Time			8	24	16	32	40
Submission Time			300.256	219.728	316.432	251.904	292.08
Update Status			300.256	219.728	316.432	251.904	292.08
Start Execution time			300.256	219.728	316.432	251.904	292.08
Execution Time			292.912	519.472	492.245	624.56	804.421
Turn around time			593.168	739.2	808.677	876.464	1096.501
Moving Cost (£)			5	5	5	5	5

Graph 6-3 express how turnaround time is decreasing in dynamic scheduler when we have 2 Processing element (PE) and in static recovery schemes has same results.



Graph 6-3: Crash and turnaround time of 7 computers (Dynamic Scheduler with different PE)

Therefore, we can easily depict that in some cases static recovey schemes have better turn around time and in some cases dynamic scheduler work better. The worst case of Golomb and Trapezium is traced when we have the compputer or clusters have less capacity. In that case crashing any machine the load will transfer to another machine / computer but the cmputer will not take that job due to less capability and job will move from here to there and it will not finish, this is the worst case when job will not execute and resouces will wait for job.

6.3.5 Comparison with static and Dynamic with different processing element

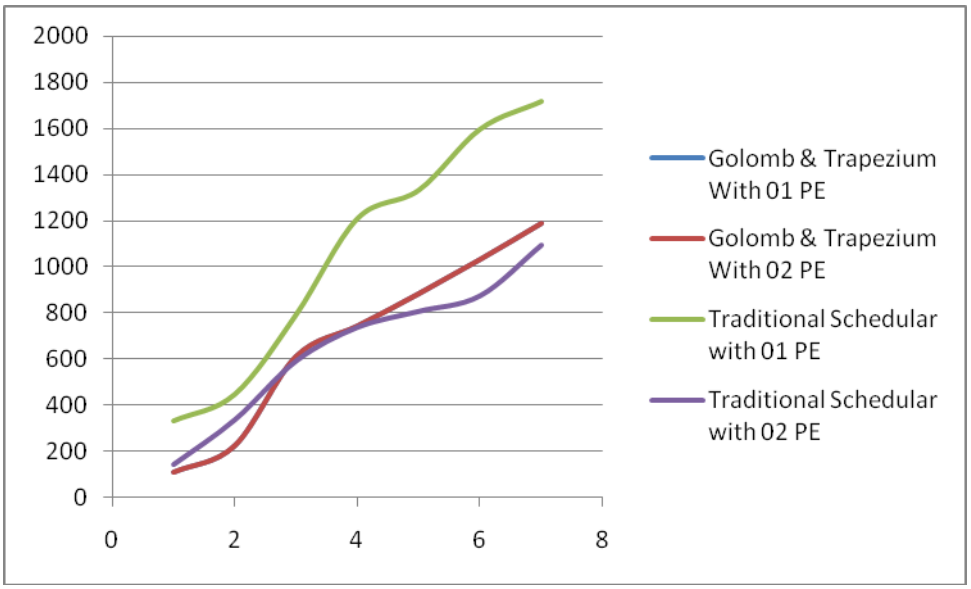
Another interesting observation is when we assume different number of crashed computers. Here we simulate both scenarios when first only one computer crashes, than two computers and so on. In the case when we have only pne processing element static scenario is much better than dynamic. However, when we have two processing elements, the dynami scenario is better than static. The interesting observation is that for two processing elements static scenario has the same results as for one processing element.
Why?

In table 6-4 we present these results looking at the trun around time.

Table 6-4: Comparison with static and Dynamic with different processing element

Computers/Schemes	Golomb & Trapezium with 01 PE	Golomb & Trapezium with 02 PE	Traditional Scheduler with 01 PE	Traditional Scheduler with 02 PE
Comp 0	111	111	331.709	142.552
Comp 1	227.064	227.064	447.773	337.469
Comp 2	613.328	613.328	793.044	593.168
Comp 3	744.504	744.504	1209.22	739.2
Comp 4	884.68	884.68	1332.12	808.677
Comp 5	1032.856	1032.856	1596.472	876.464
Comp 6	1188.944	1188.944	1716.56	1096.501

From graph 6-4 we can see the turnaround time of static and dynamic recovery schemes. However, as we have mentioned above, the total turnaround time for static scenario is same in case when we increase the number of processing elements, therefore the blue line is not visible.



Graph 6-4: Static and Dynamic Recovery scheme with different processing element

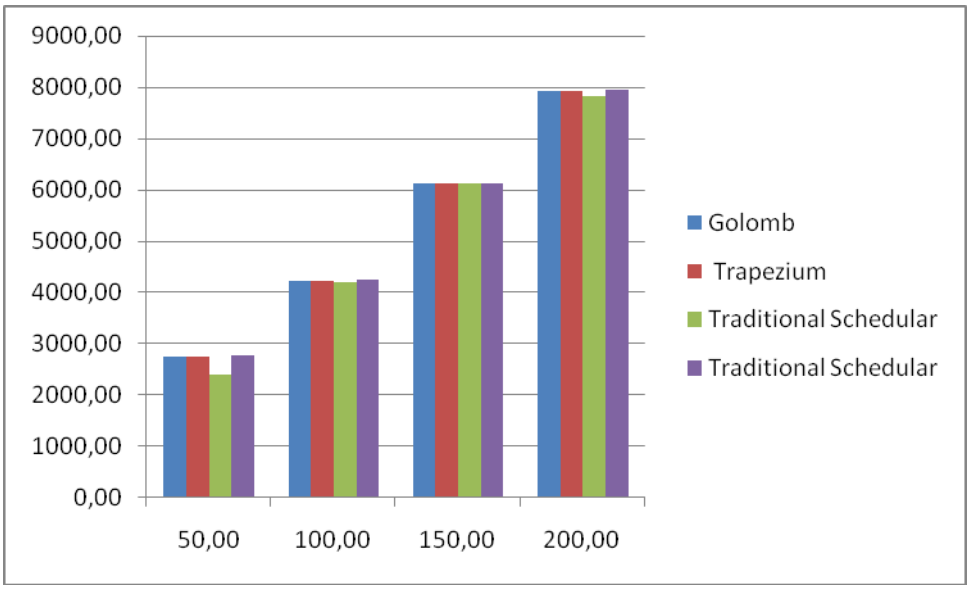
6.3.6 Analysis (Simulation results)

In our experiment we have also done the simulation of larger number of computers from 50 until 200 computers in a cluster. Here, we present the results (the turnaround time) with large number of computers having Golomb, Trapezium and dynamic recovery schemes. We depict data from table 6-5 where we have described results from 50 to 200 numbers of computers. Graph 6-5 is the same reflection of the table 6-5.

Table 6-5: Turnaround time of Dynamic and Static Recovery Schemes

Total Computers	Golomb (Seconds)	Trapezium (Seconds)	Traditional Scheduler (Seconds)	Traditional Scheduler (Seconds)
50.00	2755.49	2749.93	2391.75	2784.43
100.00	4231.69	4233.88	4206.93	4241.63
150.00	6124.59	6119.50	6126.14	6126.14
200.00	7922.00	7919.99	7835.56	7939.40

Graphical representation of table is given below:



Graph 6-5: Turnaround time of Dynamic and Static Recovery Schemes

Finally, we can analyze from the results given in the graph 6-5 that for more number of computers the turnaround time is almost same for all scenario, i.e. the load distributes evenly among resources for static and dynamic scenario.

7 RESULTS

The primary objective of this thesis is to evaluate how gird computing works with their infrastructure and in which cases dynamic scheduler is preferable. How can static recovery schemes play an effective role in large distributed system when load balancing is a key element? How can we get optimality in maximum number of crash nodes using dynamic or static recovery schemes?

To achieve the results we have developed two modules, one for Golomb and other for Trapezium. C++ programming language has been used for the implementation of module. With the help of those modules we can produce n recovery schemes for Golomb and Trapezium and it's a good contribution for further researchers who can get the n recovery schemes and can do simulation or emulation or can check on grid as well. We can compute n recovery schemes sequences for n crash computers but the optimal is just 24 for Golomb.

The construction of Golomb and Trapezium is explained theoretically in Chapter 2. How Golomb and Trapezium sequences constructed by our module is explained in appendix D.

After the construction of Golomb and Trapezium module we have developed three schedulers: one for dynamic recovery schemes and 2 for static recovery schemes. After construction of schedulers we did simulation and we compared how static recovery schemes play an effective role in grid computing.

We have computed actual CPU time, turnaround time and migration time when any task is completed with and without any failure of node. We have used Gridsim tool that provide facility of simulation for Gridsim entities like resources, broker, Gridlet and networks.

We have computed results from 3 to 200 numbers of computers. In empirical analysis examples have been given of 7 crash computers.5 out of 7crashed explicitly (results are given in Chapter 6.

As we explained earlier that Gridsim toolkit has been used for simulation of static and dynamic recovery schemes. Scope here is limited, like total number of users, total number of machines, total number of processing elements, network topology, network speed and traffic etc.

Hence, from 3 to 20 number of crash computers we have analyzed that static recovery schemes work better than dynamic recovery schemes, but as we increase the number of computers and the processing element or machines in simulation then dynamic recovery schemes provide better results. This is again a limitation of our scope. we are doing simulation for n number of crash computers when we have one processor in each machine. We are not dealing parallelism in static recovery schemes.

Finally, we come to the conclusion that static recovery schemes are better than dynamic recovery schemes, but it can be worse when static recovery schemes takes the computer with less capability and the jobs are moved from one to another and is never complete.

8 DISCUSSION/ANALYSIS

In many distributed or cluster system, designer must implement a recovery scheme which identify how the load should be distributed when one or more computers crash. The objective of recovery scheme is to keep the load as evenly distributed as possible, even in worst case, when most unfavorable combination of computers crash. We tried to analyze and compare performance of Golomb, Trapezium and dynamic recovery scheme.

In our research n identical computers have been assumed, in each computer one or parallel processes / tasks / Gridlet are executed. Static recovery schemes like Golomb and Trapezium provides guarantee of optimality in worst case of resource failure. Our contribution in this paper is that we have developed two modules for Golomb and Trapezium recovery schemes in C++ programming language, and then we have compared the performance of Golomb and Trapezium recovery scheme with dynamic recovery scheme by using GridSim simulator.

Regarding our comparative analysis between Golomb and Trapezium, it has been proved that Trapezium recovery scheme is comparatively better than Golomb recovery scheme, and it's easy to compute. Trapezium recovery scheme provide better performance as compare to Golomb, when number of computers is large. There is no difference of performance between Golomb and Trapezium when there is small number of computers. We have analyzed that Golomb and Trapezium recovery schemes provide same result when the number of computers is less than 425. If numbers of computers are greater than 425 then Trapezium recovery schemes provide better results as compare to the Golomb recovery schemes.

In empirical analysis we depicted the scenario when we have 7 computers with 1 processing element (PE). If 5 computers crash then the total turnaround time is 1188.944 seconds for the static scenario. In that scenario, we have also analyzed that the results are same if we increase processing element from 1 to 2. The results are same because we are using the same recovery scheme for both processing element. In theory there should be more then one recovery scheme when we have heterogeneous processor.

We have also analyzed the same scenario with 7 computers with dynamic scheduler. Here, the total turnaround time is 1716.56 seconds, which is greater as compare to static recovery scheme. In that scenario, if we increase processing element (from 1 to 2) then dynamic scheduler work better than static, because the total turnaround time of 7 tasks / Gridlet is 1096.501seconds. On the behalf of these results we can approve that in some cases dynamic scheduler give better results (better turnaround time) and in others Golomb and Trapezium. It depends on scenario, if processing elements are more i.e. 2 to n, then dynamic scheduler provide better results as compare to Golomb and Trapezium.

To get the exact result for Golomb and Trapezium with more than one processing element one should have as many recovery schemes for individual processing element or machine. This can be a future work.

We have analyzed that Golomb and Trapezium provide better turnaround time as compare to dynamic scheduler when we have 50 computers and one processing element. In this scenario, Golomb recovery scheme takes total turnaround time 2755 seconds, and Trapezium turnaround time 2750 seconds, while dynamic scheduler turnaround time is 2784 seconds. In this scenario, if we increase processing element in dynamic scheduler then the total turnaround time is 2391 seconds. This shows that if dynamic scheduler uses two processing elements then it gives better results as compare to Golomb and Trapezium.

As we have discussed earlier the worst case of static recovery schemes are few, one of them is, when we have same capacity / capability of machines or nodes. In the case of crashing of any machine the load will transfer to another machine but the computer will not take that job due to less capability / capacity and thus job will move from here to there and it will not finish, this is the worst case when job will not execute and resources will wait for job.

8.1 Future Work

The future work requires analysis, when we have heterogonous computers, i.e. computers with different numbers of processors. When there are different capacities / weights of job with diverse network speed.

SUMMARY

The basic purpose of this thesis is to investigate the behavior of grid computing with respect to load balance. We studied theoretically Golomb and Trapezium recovery schemes, which are the static recovery schemes and analyzed how these static schemes work in a grid scenario. We also analyzed how dynamic scheduler handles and balance the load in the same scenario. After construction of two modules in C++ programming language we have generated recovery schemes for n number of crash computers. Gridsim tool has been used for simulations of two static and one dynamic scheduling.

From the results of simulations we conclude that:

- Golomb and Trapezium have been proved best overall as compare to dynamic schedulers, when we have one processing element for simulation.
- Dynamic is better when the static recovery scheme select the less capability of computers

REFERENCES

1. Adarsh's Grid Home, Grid Computing, Grid in Need is a Grid Indeed. http://www.adarshpatil.com/newsite/grid.htm [Last access on February 24, 2008]
2. Browne, J. C., Grid computing as applied distributed computation: a graduate seminar on Internet and Grid computing, In Proceedings of the 2004 IEEE international Symposium on Cluster Computing and the Grid (April 19 - 22, 2004), CCGRID. IEEE Computer Society, Washington, DC, pp. 239-245.
3. Buyya, R., Broberg, J., Assuncao, M.D., Sulistio, A., GridSim: A Grid Simulation Toolkit for Resource Modelling and Application Scheduling for Parallel and Distributed Computing, http://www.buyya.com/gridsim/ [Last access on 23 April, 2008] 3, (1833), 2005, pp. 2017-2035, (doi: 10.1098/rsta.2005.1626)
4. Buyya, R., Murshed, M., GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing, The Journal of Concurrency and Computation: Practice and Experience (CCPE), Wiley Press, 2002.
5. Cameron, D. G., Millar, A. P., Nicholson, C., Carvajal-Schiaffino, R., Stockinger, K., Zini, F., Analysis of Scheduling and Replica Optimisation Strategies for Data Grids Using OptorSim, Journal of Grid Computing, Springer Netherlands, 2004.
6. Cluster computing. Copyright © 2004 Cisco Systems. Cisco Systems. Inc. http://www.cisco.com/en/US/solutions/collateral/ns340/ns517/ns224/ns500/net_implementation_white_paper0900aecd80281846.pdf [visited on 25 April, 2008]
7. Computing at Dartmount, Getting Started with the Grid at Dartmouth, Course Handout. http://www.dartmouth.edu/~rc/classes/intro_grid/grid_overview.html [Last access on February 18, 2008]
8. Condor Introduction, Asia Pacific Grid Workshop Tokyo, 2001 http://www.cs.wisc.edu/condor [last access on 22 April, 2008]
9. Coomer, J., Chaubal, C., Introduction to the Cluster Grid – Part 1, Sun BluePrints™. OnLine—August 2002. [http://www.sun.com/blueprints]
10. Cuimin, L., Hiroyasu, T., Miki, M., Windows Cluster Step by Step, ISDL report no 20060723002, 2006. http://mikilab.doshisha.ac.jp/dia/research/report/2006/0723/002/report20060723002.html [visited on 21 May, 2008]
11. Digipede, Grid and Cluster Computing: Options for Improving Windows® Application Performance. Digipede Technologies LLC, 2003. http://download.microsoft.com/download/7/0/8/708547e7-3617-42a4-ae46-7a7fae335087/Digipede_CCS_White%20Paper.pdf [Last access on February 22, 2008]
12. Fahringer, T., Seminar work, Grid Computing 2 (SE 2.0, 703822), Simulating the grid, http://dps.uibk.ac.at/~tf/lehre/ws06/gc2/sourcen/walter.pdf [Last time access on 27 April, 2008]
13. Ferreira, L., Berstis, V., Armstrong, J., Kendzierski, M., Neukoetter, A., Takagi, M., Bing-Wo, R., Amir, A., Murakawa, R., Hernandez, O., Magowan, J., Bieberstein, N., Introduction to Grid Computing with Globus, IBM International Technical Support Organization, September 2003. [ibm.com/redbooks]
14. Figler, J., Load Balancing In Concurrent Parallel Applications. Rochester Institute of Technology Computer Engineering Department Rochester, May 1999.
15. Haddad, I., Leangsuksun, C. Stephen, L. S., HA-OSCAR: Five Steps to a High-Availability Linux Cluster, 2005. http://www.linuxdevcenter.com/pub/a/linux/2005/02/03/haoscar.html [visited on May 22, 2008].

16. Hariri, S., Parashar, S., Tools and Environments for Parallel and Distributed Computing, Publisher Wiley-IEEE, 2004, ISBN:0471332887 Science, vol. 605. Springer-Verlag, London, 295-310.
17. http://dsl-wiki.cs.uchicago.edu/index.php/Wiki:Projects#GangSim_Simulator [Last time visited on 18 April, 2008]
18. http://en.wikipedia.org/wiki/Golomb_ruler [Last time visited on 24 April, 2008]
19. http://searchcio-midmarket.techtarget.com/sDefinition/0,,sid183_gci214456,00.html (visited on 15 March, 2008)
20. Kameda, H., Fathyty, E.S., Ryu, I., Li, J., A Performance Comparison of Dynamic vs. Static Load Balancing Policies in a Mainframe -- Personal Computer Network Model, Proceedings of the 39th IEEE Conference on Dedsion and Control, 2000.
21. Klonowska, K., Theoretical aspects on performance bounds and fault tolerance in parallel computing, Blekinge Institute of Technology doctoral dissertation series no 2007:18, ISSN 1653-2090, ISBN 978-91-7295-126-6
22. Klonowska, K., Lundberg, L., and Lennerstad, H., Using Golomb Rulers for Optimal Recovery Schemes in Fault Tolerant Distributed Computing, In Proceedings of the 17th international Symposium on Parallel and Distributed Processing (April 22 - 26, 2003), IPDPS, IEEE Computer Society, Washington DC 213.
23. Klonowska, K., Lundberg, L., Lennerstad, H., Svahnberg, C., Extended Golomb Rulers as the New Recovery Schemes in Distributed Dependable Computing, In Proceedings of the 19th IEEE international Parallel and Distributed Processing Symposium (Ipdps'05) - Workshop 16 - Volume 17 (April 04 - 08, 2005), IPDPS. IEEE Computer Society, Washington, DC, 279.1. DOI= http://dx.doi.org/10.1109/IPDPS.2005.215
24. Leangsuksun, C., Shen, L., Tong, L., Song, H., Scott, S.L., Availability prediction and modeling of high mobility OSCAR cluster, Proceedings IEEE International Conference on Cluster Computing, 2003.
25. Lindh, B., Sun™ Based Beowulf Cluster. Sun Microsystems AB. Sun BluePrints™ OnLine - January 2002.
26. Marek, R. and Rahm, E., Performance Evaluation of Parallel Transaction Processing in Shared Nothing Database Systems, In Proceedings of the 4th international PARLE Conference on Parallel Architectures and Languages Europe (June 15 - 18, 1992). D. Etiemble and J. Syre, Eds. Lecture Notes In Computer
27. Neumann, D.G., Economic Models and Algorithms for Grid Systems. Institute of Information Systems and Management (IISM) Universität Karlsruhe (TH), 2007. www.iw.uni-karlsruhe.de/Publications/407_EMAGS_final.pdf
28. Open source cluster application resource. http://oscar.openclustergroup.org/ (visited on 15 March, 2008)
29. OptorSim, http://www.ohloh.net/projects/12081?ref=sample [Last time access on 18 April, 2008]
30. Osman, A., Ammar, H., Dynamic Load Balancing Strategies for Parallel Computers, International Symposium on Parallel and Distributed Computing (ISPDC), 2004.
31. Pc Magazine. http://www.pcmag.com/encyclopedia_term/0,2542,t=grid+computing&i=43962,00.asp [Last access on February 25, 2008]
32. Pfister, G. F, In Search of Clusters, the Ongoing Battle in Lowly Parallel Computing, Prentice Hall PTR, 1998, ISBN 0-13-899709-8
33. Project OGR, http://www.distributed.net/ogr/ (visited on 15 March, 2008)
34. Quetier, B., Cappello, F., A survey of Grid research tools: simulators, emulators and real life platforms. http://www.lri.fr/~quetier/presentations/imacs2005_survey_slides.pdf [visited on 25 April, 2008]

35. Rahm, E., Marek, R, Dynamic Multi-Resource Load Balancing in Parallel Database Systems. In Proceedings of the 21th international Conference on Very Large Data Bases (September 11 - 15, 1995).
36. Sulistio, A., Poduval, G., Buyya, R., Tham, C., On incorporating differentiated levels of network service into GridSim, Future Gener, Comput. Syst. 23, 4 (May. 2007), 606-615, DOI= http://dx.doi.org/10.1016/j.future.2006.10.006
37. Surfleet, C., Grid storage: The topology of choice for scalable systems? , Computerworld storage networking world online, how to plan/implement storage network, 2004. http://www.snwonline.com/plan/grid_storage_06-07-04.asp [Last access on March 28, 2008]
38. Takefusa, A., Matsuoka, S., Nakada, H., Aida, K., Nagashima, U., Overview of a Performance Evaluation System for Global Computing Scheduling Algorithms. http://ninf.apgrid.org/bricks/contents/hpdc99.html [Last access on 18 April, 2008]
39. Thain, D., Tannenbaum, T., Livny, M., Distributed computing in practice: the Condor experience: Research Articles. Concurr. Comput. : Pract. Exper. 17, 2-4 (Feb. 2005), pp. 323-356. DOI= http://dx.doi.org/10.1002/cpe.v17:2/4
40. Woods, J. C., Hong, M.N., Johnston, S., Murdock, S. E., Wu, B., Tai, K., Fangohr, H., Jeffreys, P., Cox, S., Jeremy G.F., Sansom, S.P., Jonathan W.E., Grid computing and biomolecular simulation. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 36
41. Yang C.T., Lai, C.L., Apply cluster and grid computing on parallel 3D rendering. Multimedia and Expo, IEEE International Conference on Volume 2. Issue , 30-30 June 2004, Vol.2 Digital Object Identifier 10.1109/ICME.2004.1394336, pp. 859 - 862
42. Zaki, M. J., Li, W., and Parthasarathy, S. 1997. Customized dynamic load balancing for a network of workstations. J. Parallel Distrib. Comput. 43, 2 (Jun. 1997), 156-162. DOI= http://dx.doi.org/10.1006/jpdc.1997.1339

TERMINOLOGY

Hot backup	A hot backup, also called a dynamic backup, is a backup performed on data even though it is actively accessible to users and may currently be in a state of being updated. Hot backups can provide a convenient solution in multi-user systems, because they do not require downtime, as does a conventional cold backup.
Cold backup	A cold backup, also called an offline backup, is a database backup when the database is offline and thus not accessible for updating. This is the safest way to back up because it avoids the risk of copying data that may be in the process of being updated. However, a cold backup involves downtime because users cannot use the database while it is being backed up.
Recovery order	The order in which the computers are used is referred to as the recovery order, given by a recovery list.
Failover	Failover is the capability to switch over automatically to a redundant or standby computer server, system, or network upon the failure or abnormal termination of the previously active server, system, or network. Failover happens without human intervention and generally without warning, unlike switchover.
Load balancing	In computer networking, load balancing is a technique to spread work between two or more computers, network links, CPUs, hard drives, or other resources, in order to get optimal resource utilization, throughput, or response time.
Nix	Nix, is a safe and flexible deployment system providing mechanisms that can be used to define a great variety of deployment policies. The main problem with Windows from day one is that sharing violation problem, you never see that on _any_ *nix machines.
Fault	An abnormal condition or defect at the component, equipment, or sub-system level which may lead to a failure

APPENDIX 1: OPTIMAL SEQUENCE

The below table consist of all known Golomb rulers and this table is complete up to 24 orders: [31]

Order	Length	Marks
1	0	0
2	1	0 1
3	3	0 1 3
4	6	0 1 4 6
5	11	0 1 4 9 11 0 2 7 8 11
6	17	0 1 4 10 12 17 0 1 4 10 15 17 0 1 8 11 13 17 0 1 8 12 14 17
7	25	0 1 4 10 18 23 25 0 1 7 11 20 23 25 0 1 11 16 19 23 25 0 2 3 10 16 21 25 0 2 7 13 21 22 25
8	34	0 1 4 9 15 22 32 34
9	44	0 1 5 12 25 27 35 41 44
10	55	0 1 6 10 23 26 34 41 53 55
11	72	0 1 4 13 28 33 47 54 64 70 72 0 1 9 19 24 31 52 56 58 69 72
12	85	0 2 6 24 29 40 43 55 68 75 76 85
13	106	0 2 5 25 37 43 59 70 85 89 98 99 106
14	127	0 4 6 20 35 52 59 77 78 86 89 99 122 127
15	151	0 4 20 30 57 59 62 76 100 111 123 136 144 145 151
16	177	0 1 4 11 26 32 56 68 76 115 117 134 150 163 168 177
17	199	0 5 7 17 52 56 67 80 81 100 122 138 159 165 168 191 199
18	216	0 2 10 22 53 56 82 83 89 98 130 148 153 167 188 192 205 216
19	246	0 1 6 25 32 72 100 108 120 130 153 169 187 190 204 231 233 242 246
20	283	0 1 8 11 68 77 94 116 121 156 158 179 194 208 212 228 240 253 259 283
21	333	0 2 24 56 77 82 83 95 129 144 179 186 195 255 265 285 293 296 310 329 333
22	356	0 1 9 14 43 70 106 122 124 128 159 179 204 223 253 263 270 291 330 341 353 356
23	372	0 3 7 17 61 66 91 99 114 159 171 199 200 226 235 246 277 316 329 348 350 366 372
24	425	0 9 33 37 38 97 122 129 140 142 152 191 205 208 252 278 286 326 332 353 368 384 403 425

Table: Known Optimal Golomb Ruler [31]

Prediction about Optimal Golomb ruler for order 25 is written below: [31]

Order	length	Marks
25	480	0 12 29 39 72 91 146 157 160 161 166 191 207 214 258 290 316 354 372 394 396 431 459 467 480

APPENDIX 2: SEQUENCE OF GOLOMB RULERS

Golomb Rulers

1 2 4 5 8 10 14 21 15 16 26 25
3 6 9 13 18 24 35 36 31 42 51
7 11 17 23 32 45 50 52 57 67
12 19 27 37 53 60 66 78 82
20 29 41 58 68 76 92 103
30 43 62 73 84 102 117
44 64 77 89 110 127
65 79 93 115 135
80 95 119 140
96 121 144
122 146
147

Golomb Rulers

1 3 2 7 8 10 14 19 16 22 28 34
4 5 9 15 18 24 33 35 38 50 62
6 12 17 25 32 43 49 57 66 84
13 20 27 39 51 59 71 85 100
21 30 41 58 67 81 99 119
31 44 60 74 89 109 133
45 63 76 96 117 143
64 79 98 124 151
80 101 126 158
102 129 160
130 163
164

Golomb Rulers

1 4 2 8 3 9 16 19 21 24 29 31
5 6 10 11 12 25 35 40 45 53 60
7 14 13 20 28 44 56 64 74 84
15 17 22 36 47 65 80 93 105
18 26 38 55 68 89 109 124
27 42 57 76 92 118 140
43 61 78 100 121 149
62 82 102 129 152
83 106 131 160
107 135 162
136 166
167

APPENDIX 3: SEQUENCE OF TRAPEZIUM

1 13 2 3 4 17 8 12 30 27 6 45 10 58 12 11 5 28 8 2 19 20 15 1
14 15 5 7 21 25 20 42 57 33 51 55 68 70 23 16 33 36 10 21 39 35 16
16 18 9 24 29 37 50 69 63 78 61 113 80 81 28 44 41 38 29 41 54 36
19 22 26 32 41 67 77 75 108 88 119 125 91 86 56 52 43 57 49 56 55
23 39 34 44 71 94 83 120 118 146 131 136 96 114 64 54 62 77 64 57
40 47 46 74 98 100 128 130 176 158 142 141 124 122 66 73 82 92 65
48 59 76 101 104 145 138 188 188 169 147 169 132 124 85 93 97 93
60 89 103 107 149 155 196 200 199 174 175 177 134 143 105 108 98
90 116 109 152 159 213 208 211 204 202 183 179 153 163 120 109
117 122 154 162 217 225 219 216 232 210 185 198 173 178 121
123 167 164 220 229 236 224 244 240 212 204 218 188 179
168 177 222 232 240 241 252 252 242 231 224 233 189
178 235 234 243 245 269 260 254 261 251 239 234
236 247 245 248 273 277 262 273 281 266 240
248 258 250 276 281 279 281 293 296 267
259 263 278 284 283 298 301 308 297
264 291 286 286 302 318 316 309
292 299 288 305 322 333 317
300 301 307 325 337 334
302 320 327 340 338
321 340 342 341
341 355 343
356 356
357

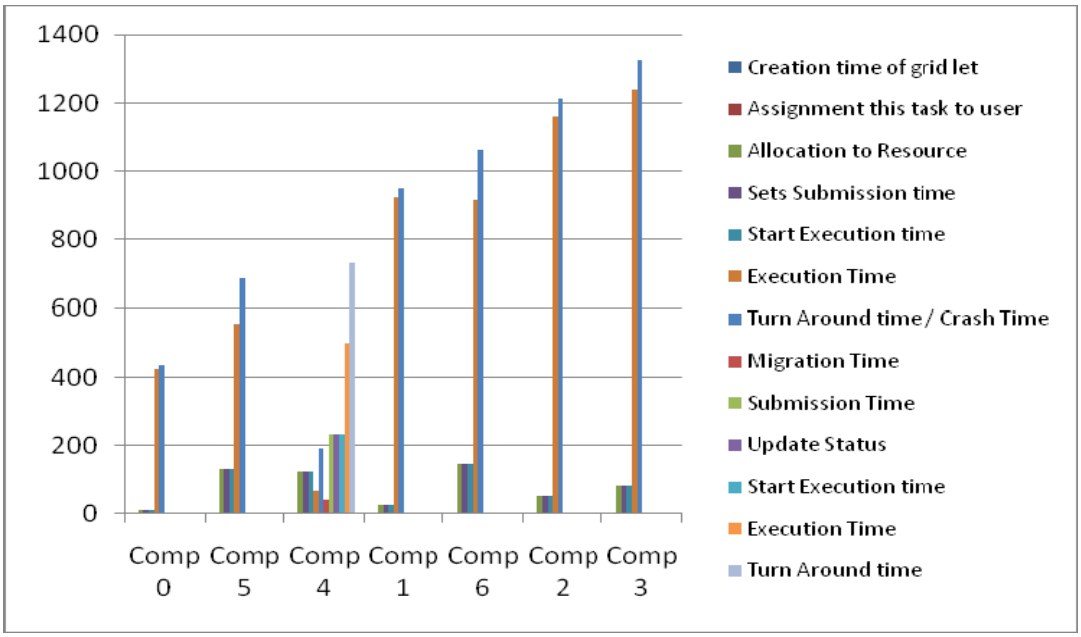
1 13 2 3 4 17 8 12 30 27 6 45 10 58 12 11 5 28 8 2 19 20 15 2
14 15 5 7 21 25 20 42 57 33 51 55 68 70 23 16 33 36 10 21 39 35 17
16 18 9 24 29 37 50 69 63 78 61 113 80 81 28 44 41 38 29 41 54 37
19 22 26 32 41 67 77 75 108 88 119 125 91 86 56 52 43 57 49 56 56
23 39 34 44 71 94 83 120 118 146 131 136 96 114 64 54 62 77 64 58
40 47 46 74 98 100 128 130 176 158 142 141 124 122 66 73 82 92 66
48 59 76 101 104 145 138 188 188 169 147 169 132 124 85 93 97 94
60 89 103 107 149 155 196 200 199 174 175 177 134 143 105 108 99
90 116 109 152 159 213 208 211 204 202 183 179 153 163 120 110
117 122 154 162 217 225 219 216 232 210 185 198 173 178 122
123 167 164 220 229 236 224 244 240 212 204 218 188 180
168 177 222 232 240 241 252 252 242 231 224 233 190
178 235 234 243 245 269 260 254 261 251 239 235
236 247 245 248 273 277 262 273 281 266 241
248 258 250 276 281 279 281 293 296 268
259 263 278 284 283 298 301 308 298
264 291 286 286 302 318 316 310
292 299 288 305 322 333 318
300 301 307 325 337 335
302 320 327 340 339
321 340 342 342
341 355 344
356 357
358

APPENDIX 4: DYNAMIC Vs STATIC

When one computer crash in Dynamic Scheme.

Description	Comp 0	Comp 5	Comp 4	Comp 1	Comp 6	Comp 2	Comp 3
Creation time of grid let	0	0	0	0	0	0	0
Assignment this task to user	0	0	0	0	0	0	0
Allocation to Resource	11	131.32	123.256	27.064	147.384	51.128	83.192
Sets Submission time	11	131.32	123.256	27.064	147.384	51.128	83.192
Start Execution time	11	131.32	123.256	27.064	147.384	51.128	83.192
Execution Time	424.297	557.041	69.304	923.102	914.86	1160.461	1240.085
Turn Around time / Crash Time	435.297	688.361	192.56	950.166	1062.244	1211.589	1323.277
Migration Time			40				
Submission Time			232.56				
Update Status			232.56				
Start Execution time			232.56				
Execution Time			500				
Turn Around time			732.56				
Moving Cost (£)			5				

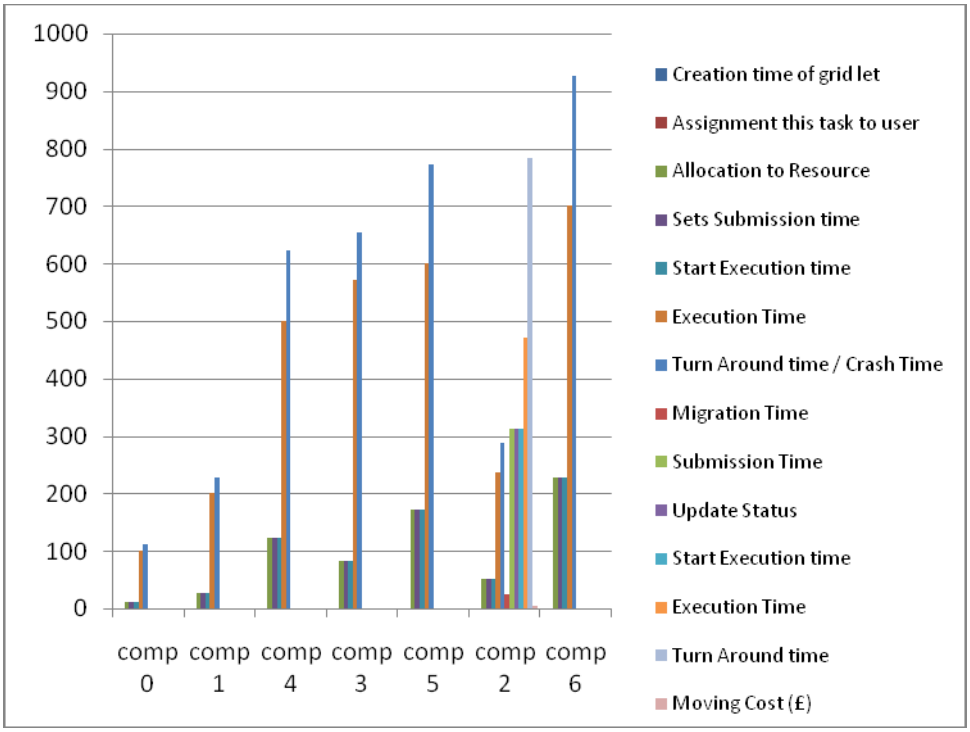
The graphical representation of the above table can also be seen in graph below



When one computer crash in Static Scheme.

Description	0	1	4	3	5	2	6
Creation time of grid let	0	0	0	0	0	0	0
Assignment this task to user	0	0	0	0	0	0	0
Allocation to Resource	11	27.064	123.256	83.192	171.32	51.128	227.384
Sets Submission time	11	27.064	123.256	83.192	171.32	51.128	227.384
Start Execution time	11	27.064	123.256	83.192	171.32	51.128	227.384
Execution Time	100	200	500	570.952	600	237.112	700
Turn Around time / Crash Time	111	227.064	623.256	654.144	771.32	288.24	927.384
Migration Time						24	
Submission Time						312.24	
Update Status						312.24	
Start Execution time						312.24	
Execution Time						470.95	
Turn Around time						783.192	
Moving Cost (£)						5	

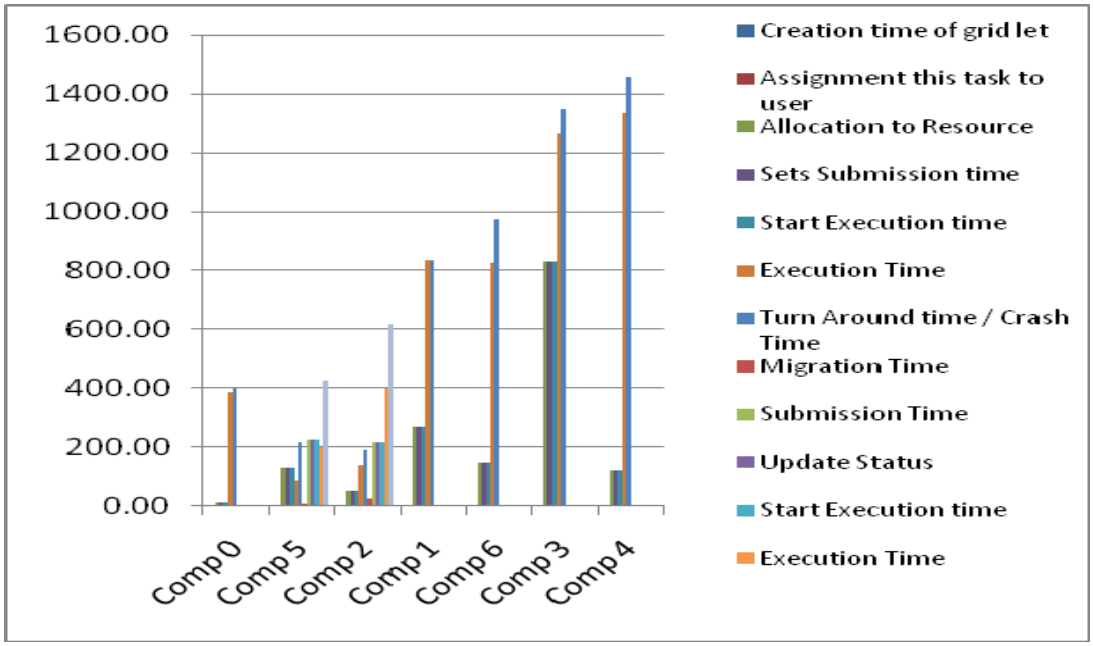
The graphical representation of the above table can also be seen in graph below



When two computers crash in Dynamic Scheme

Description	Comp 0	Comp 5	Comp 2	Comp 1	Comp 6	Comp 3	Comp 4
Creation time of grid let	0.00	0.00	0	0.00	0.00	0.00	0.00
Assignment this task to user	0.00	0.00	0	0.00	0.00	0.00	0.00
Allocation to Resource	11.00	131.19	51	270.64	147.19	831.28	123.13
Sets Submission time	11.00	131.19	51	270.64	147.19	831.28	123.13
Start Execution time	11.00	131.19	51	270.64	147.19	831.28	123.13
Execution Time	387.97	85.35	141.30	836.16	826.94	1263.44	1333.94
Turn Around time / Crash Time	398.97	216.54	192.37	836.16	974.13	1346.57	1457.07
Migration Time		8.00	24.00				
Submission Time		224.54	216.37				
Update Status		224.54	216.37				
Start Execution time		224.54	216.37				
Execution Time		200.00	400.00				
Turn Around time		424.54	616.37				
Moving Cost (£)		5	5				

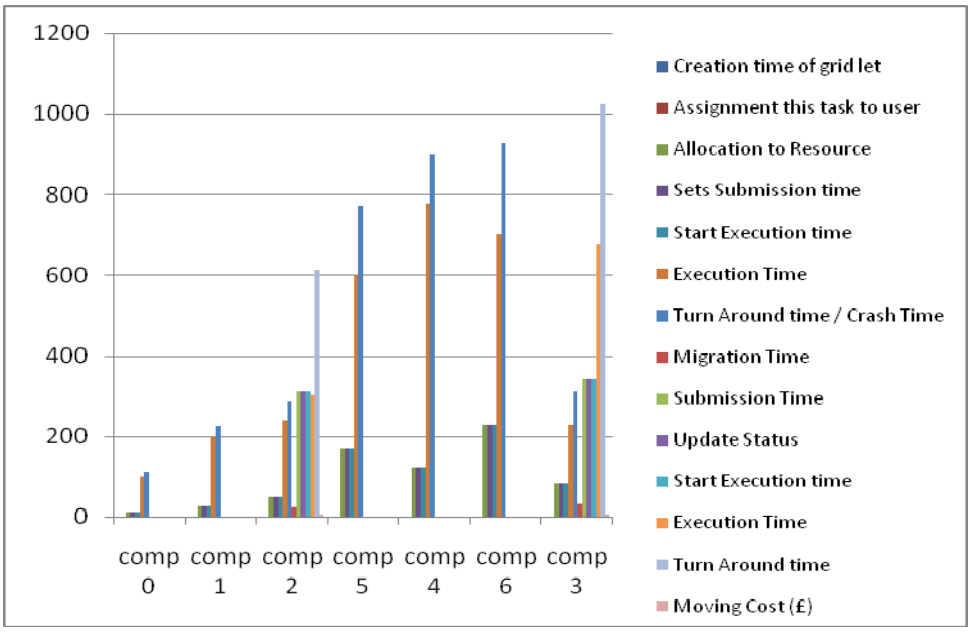
The graphical representation of the above table can also be seen in graph below



Description	comp 0	comp 1	comp 2	comp 5	comp 4	comp 6	comp 3
Creation time of grid let	0	0	0	0	0	0	0
Assignment this task to user	0	0	0	0	0	0	0
Allocation to Resource	11	27.064	51.128	171.32	123.256	227.384	83.192
Sets Submission time	11	27.064	51.128	171.32	123.256	227.384	83.192
Start Execution time	11	27.064	51.128	171.32	123.256	227.384	83.192
Execution Time	100	200	237.112	600	778.84	700	229.22
Turn Around time / Crash Time	111	227.064	288.24	771.32	902.096	927.384	312.416
Migration Time			24				32
Submission Time			312.24				344.416
Update Status			312.24				344.416
Start Execution time			312.24				344.416
Execution Time			301.088				678.84
Turn Around time			613.328				1023.256
Moving Cost (£)			5				5

When two computers crash in Static Scheme

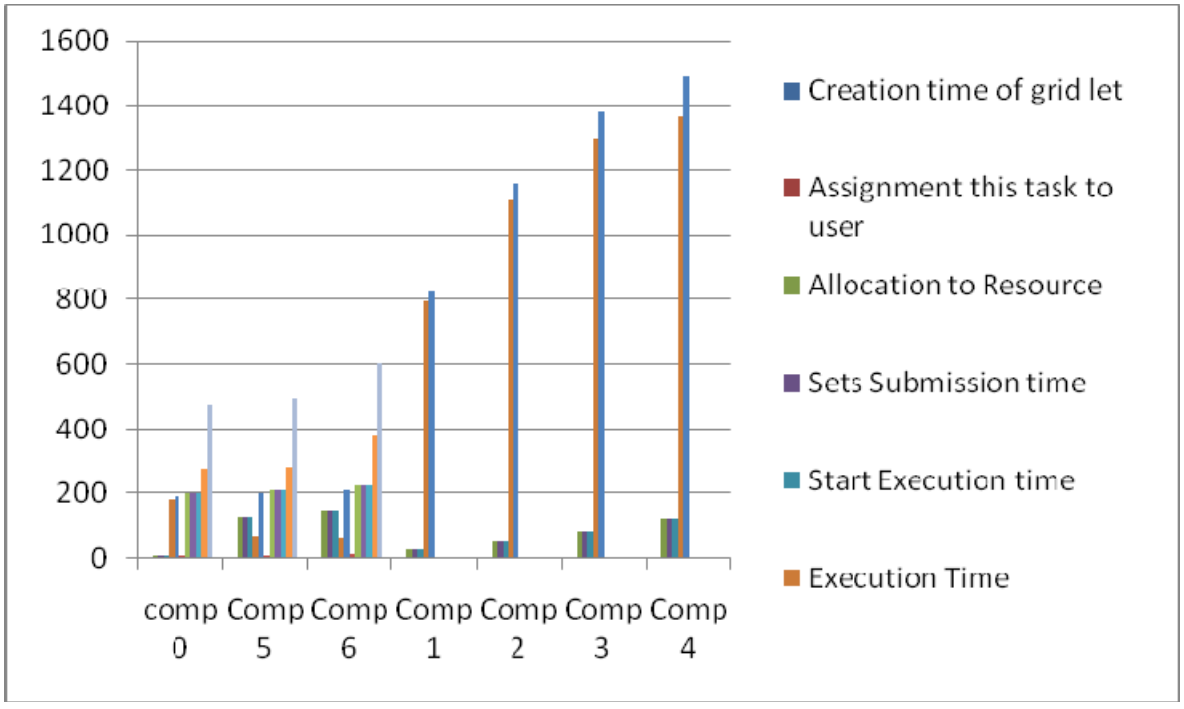
The graphical representation of the above table can also be seen in graph below



When three computers crash in Dynamic Scheme

Description	comp 0	Comp 5	Comp 6	Comp 1	Comp 2	Comp 3	Comp 4
Creation time of grid let	0	0	0	0	0	0	0
Assignment this task to user	0	0	0	0	0	0	0
Allocation to Resource	11	131.32	147.384	27.064	51.128	83.192	123.256
Sets Submission time	11	131.32	147.384	27.064	51.128	83.192	123.256
Start Execution time	11	131.32	147.384	27.064	51.128	83.192	123.256
Execution Time	181.56	69.416	61.528	795.937	1107.969	1297.281	1367.233
Turn Around time / Crash Time	192.56	200.736	208.912	823.001	1159.09	1380.47	1490.48
Migration Time	8	8	16				
Submission Time	200.56	208.736	224.912				
Update Status	200.56	208.736	224.912				
Start Execution time	200.56	208.736	224.912				
Execution Time	275.56	284.736	377.148				
Turn Around time	476.12	493.47	602.06				
Moving Cost (£)	5	5	5				

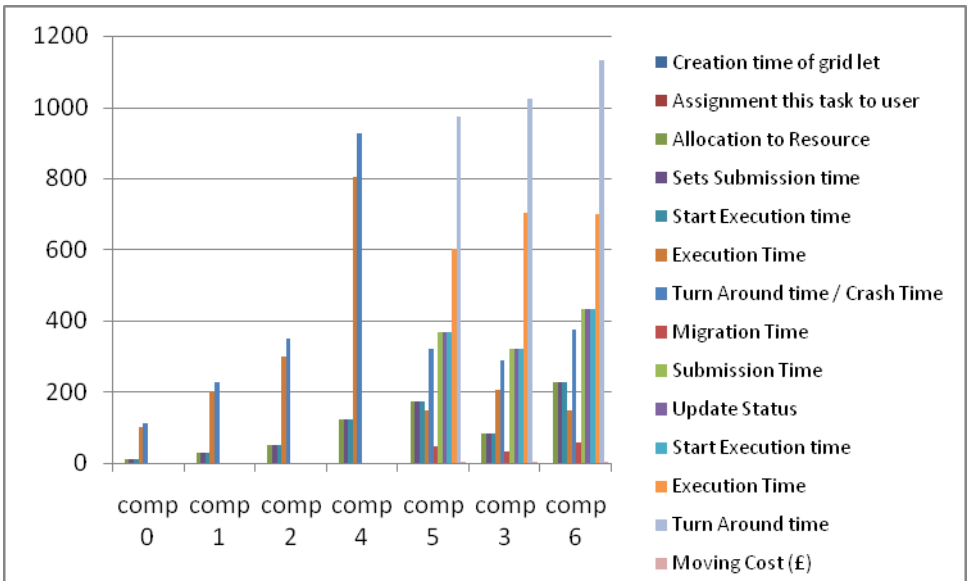
The graphical representation of the above table can also be seen in graph below



Description	comp 0	comp 1	comp 2	comp 4	comp 5	comp 3	comp 6
Creation time of grid let	0	0	0	0	0	0	0
Assignment this task to user	0	0	0	0	0	0	0
Allocation to Resource	11	27.064	51.128	123.256	171.32	83.192	227.384
Sets Submission time	11	27.064	51.128	123.256	171.32	83.192	227.384
Start Execution time	11	27.064	51.128	123.256	171.32	83.192	227.384
Execution Time	100	200	300	803.016	149.096	205.048	147.92
Turn Around time / Crash Time	111	227.064	351.128	926.272	320.416	288.24	375.304
Migration Time					48	32	56
Submission Time					368.416	320.24	431.304
Update Status					368.416	320.24	431.304
Start Execution time					368.416	320.24	431.304
Execution Time					603.444	703.016	700
Turn Around time					971.86	1023.256	1131.304
Moving Cost (£)					5	5	5

When three computers crash in Static Scheme

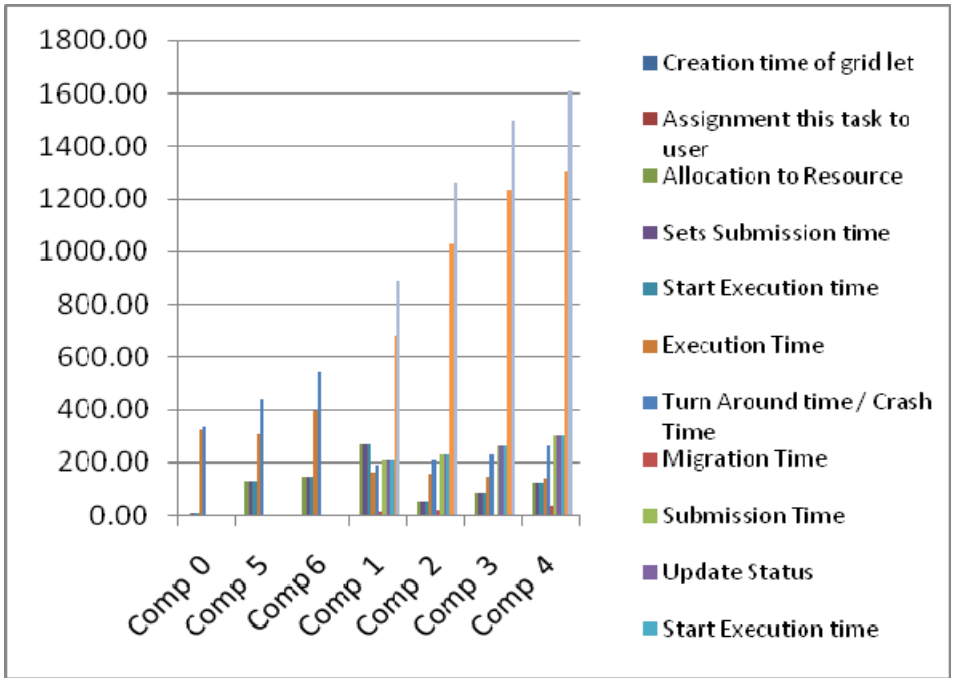
The graphical representation of the above table can also be seen in graph below



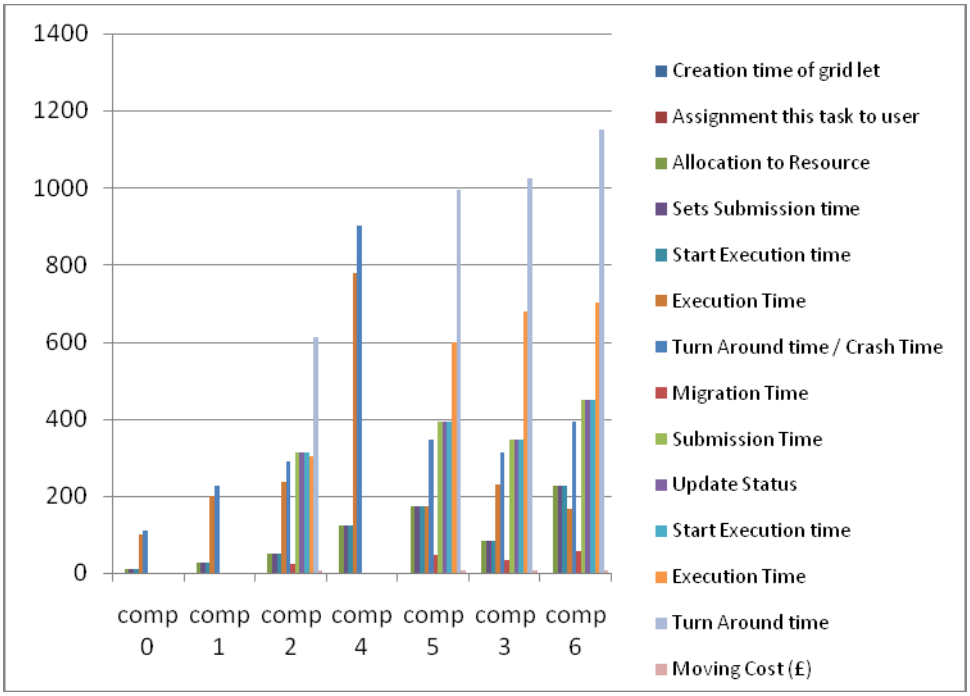
When four computers crash in Dynamic Scheme

Description	Comp 0	Comp 5	Comp 6	Comp 1	Comp 2	Comp 3	Comp 4
Creation time of grid let	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Assignment this task to user	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Allocation to Resource	11.00	131.19	147.19	270.64	51.06	83.13	123.13
Sets Submission time	11.00	131.19	147.19	270.64	51.06	83.13	123.13
Start Execution time	11.00	131.19	147.19	270.64	51.06	83.13	123.13
Execution Time	329.72	309.59	396.25	165.30	157.48	149.59	141.77
Turn Around time / Crash Time	340.72	440.78	543.44	192.37	208.54	232.72	264.90
Migration Time				16.00	24.00		40.00
Submission Time				208.37	232.54	264.72	304.90
Update Status				208.37	232.54	264.72	304.90
Start Execution time				208.37	232.54	264.72	304.90
Execution Time				681.90	1030.26	1231.26	1303.97
Turn Around time				890.27	1262.8	1495.76	1608.88
Moving Cost (£)				5	5	5	5

The graphical representation of the above table can also be seen in graph below



When four computers crash in Static Scheme

Description	Comp 0	Comp 1	Comp 2	Comp 4	Comp 5	Comp 3	Comp 6
Creation time of grid let	0	0	0	0	0	0	0
Assignment this task to user	0	0	0	0	0	0	0
Allocation to Resource	11	27.064	51.128	123.256	171.32	83.192	227.384
Sets Submission time	11	27.064	51.128	123.256	171.32	83.192	227.384
Start Execution time	11	27.064	51.128	123.256	171.32	83.192	227.384
Execution Time	100	200	237.112	778.84	173.272	229.224	165.384
Turn Around time / Crash Time	111	227.064	288.24	902.096	344.592	312.416	392.768
Migration Time			24		48	32	56
Submission Time			312.24		392.592	344.416	448.768
Update Status			312.24		392.592	344.416	448.768
Start Execution time			312.24		392.592	344.416	448.768
Execution Time			301.088		600.088	678.84	701
Turn Around time			613.328		992.68	1023.25	1149.76
Moving Cost (£)			5		5	5	5

The graphical representation of the above table can also be seen in graph below