AN IMPROVED JOB SCHEDULING ALGORITHM IN GRID COMPUTING ENVIRONMENT USING FAULT TOLERANCE MECHANISMS

BY

HAJARA IDRIS

DEPARTMENT OF MATHEMATICS

AHMADU BELLO UNIVERSITY, ZARIA

NIGERIA

AUGUST, 2015.

AN IMPROVED JOB SCHEDULING ALGORITHM IN GRID COMPUTING ENVIRONMENT USING FAULT TOLERANCE MECHANISMS

BY

Hajara IDRIS

MSc/SCI/13071/2011-12

A DISSERTATION SUBMITTED TO THE SCHOOL OF POSTGRADUATE STUDIES,

AHMADU BELLO UNIVERSITY, ZARIA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF A MASTER OF SCIENCE DEGREE IN COMPUTER SCIENCE

DEPARTMENT OF MATHEMATICS,

FACULTY OF SCIENCE

AHMADU BELLO UNIVERSITY, ZARIA

NIGERIA

AUGUST, 2015

DECLARATION

I declare that the work in this thesis entitled AN IMPROVED JOB SCHEDULING ALGORITHM IN GRID COMPUTING ENVIRONMENT USING FAULT TOLERANCE MECHANISMShas been performed by me in the Department of Mathematics under the supervision of Professor S.B Junaidu and Dr. S.E Abdullahi. The information derived from the literature has been duly acknowledged in the text and list of references provided. No part of this project report was previously presented for another degree or diploma at any university.

Hajara IDRIS		
Name of student	Signature	Date

CERTIFICATION

This thesis entitled AN IMPROVED JOB SCHEDULING ALGORITHM IN GRID COMPUTING ENVIRONMENT USING FAULT TOLERANCE MECHANISMSby IDRIS HAJARA meets the regulation governing the award of the degree of M.Sc Computer Science of Ahmadu Bello University, and is approved for its contribution to knowledge and literary presentation.

(Signature) _____

_____ Date _____

Chairman, Supervisory Committee

(Signature) _____

_____ Date _____

Member, Supervisory Committee

(Signature) _____

_____ Date _____

Member, Supervisory Committee

(Signature) _____

_____ Date _____

Head of Department

(Signature) _____

_____ Date _____

Dean, School of Postgraduate Studies

DEDICATION

ALHAMDULILLAHI, to ALLAH be the Glory, I dedicate this research work to my family, who have been a constant source of love, concern, support and strength all these years.None of this would have been possible without the love andpatience of my entire family.

ACKNOWLEDGEMENT

Thanks are probably intricate than one may think, for the simple reason that it isimpossible to write in a few lines all the people who have contributed to achieving thisimportant goal of my life (but they know it).

My deepest gratitude goes to ALMIGHTY ALLAH for His infinite blessings, whose mercy has been keeping me alive till this day and in sound health. Alhamdulillah, all praises belong to Allah for His blessings and protection, indeed I am deeply indebted to Him, without which my work would not have been possible.

My sincere gratitude goes to Professor S.B Junaidu (Major Supervisor) and Dr. S.E Abdullahi (Minor Supervisor), for their guidance, ideas, valuable direction and support throughout this research work. They are the brain behind this research work; this work would never have been possible without these supervisors. Their insights and suggestions on the research work and for valuable supervision at various stages of my work has helped me in many ways and has developed my academic thinking, problem solving and technical writing which will be very helpful in my future work and research.I also acknowledge the effort of the entire lecturers of the Department of Mathematics, Ahmadu Bello University Zaria, for their academic and intellectual support towards the success of this dissertation.

It is agreat pleasure to take the opportunity to thank the rest of my research colleagueswhich I have come across over the years, past and present colleagues at the department and other department for all work related and unrelated discussions.

I am indeed grateful to a great deal of people, which by just mentioning names the length of this page will be insufficient. All I can say is May Almighty Allah continue to reward and bless their efforts.

Most importantly, I thank the ones I hold dearly**MY FAMILY**. Without your constant encouragement, un-conditional love,psychological and economical support, I would not have been able to undertake this endeavor.

I am sure that the outcome would not have been nearly as good as it turned out without their help and support. I would like to express my heart-felt gratitude to them.May Almighty Allah bless you all.

ABSTRACT

Grid scheduler, schedules user jobs on the best available resource in terms of resource characteristics by optimizing time, and resource failure in grid is no longer an exception but a regular event. Andresources are increasinglybeing used by the scientific community to solve computationallyintensive problems which typically run for days oreven months.It is therefore absolutely essential that long-running applications are able to tolerate failures and avoid computation of the task from scratch when resource failure occurred, to satisfy the user's QoS requirement.An Improved Job Scheduling Algorithm in Grid Computing Environment Using Fault Tolerance Mechanisms is proposed. The technique employed here, isthe use of resource failure rate, as well as checkpoint-based roll back recovery strategy. Checkpointing aims at reducing the amount of work that is lost upon failure of the system by intermediately saving the state of the system. A comparison of our proposed approaches with Moallem's ACO, the result shows that the proposed algorithm achieved up to 13% reduction in makespan, 12% maximization in terms of throughput and 12% maximization in ATA when the Gridlets are varied and the resources are kept constant. Also when the Resources are varied and Gridlets are kept constant, the proposed algorithm achieved 18% reduction in makespan, 18% maximization in terms of throughput and up to 14% maximization in ATA.

Table of Content

DECLARATION ii

CERTIFICATION iii

DEDICATION iv

ACKNOWLEDGEMENT v

LIST OF FIGURES ix

LIST OF TABLES x

CHAPTER ONE: INTRODUCTION 1

1.1 Background of the Study 1

1.1.1 General Issues in Grid System: Principles 3

1.2 Problem Statement 4

1.3 Research Question 5

1.4 Research Motivation 6

1.5 Research Aim and Objectives 6

1.6 Research Methodology 7

1.7 Scope and Limitation 7

1.8 Contribution to Knowledge 8

1.9 Dissertation Organization 8

1.10 Definition of Terms 9

CHAPTER TWO: LITERATURE REVIEW 13

2.1 Introduction 13

2.2 The Grid System 13

2.2.1 Types of grid system 13

2.2.2 Application of grid computing 15

2.2.3 Challenges to grid computing 16

2.3 Resource Management System 17

2.4 Grid Resource Scheduling 18

2.5 Artificial Intelligence (AI) 20

2.5.1 Using swarm intelligence techniques 20

2.6 Fault Tolerance 22

2.6.1 Checkpointing 25

2.7 Checkpoint Interval 31
2.8 Architecture of Fault Tolerance in Grid Computing 31
2.9 Review of Some Proposed ACO Scheduling Algorithms 33
2.10 Current Fault Tolerant Scheduling Algorithms 44
2.11 Implementation Tools 51
2.11.1 Java programming language 51
2.11.2 Gridsim toolkit 52
2.11.3 Reason for gridsim toolkit 52
CHAPTER THREE: MATERIALS AND METHODS 54
3.1 Introduction 54
3.2 The Proposed Scheduling ACO Algorithm 54
3.3 The Proposed Scheduling Strategy in Grid Environment 55
3.4 Decision on Next Resource to Visit 57
3.5 Decision Making 59
3.6 An Illustration of the Model 60
3.7 Architecture of the Proposed Fault Tolerant Technique 60
3.8 Checkpointing Tool Implementation 64
3.9 An Illustration of the Rollback Recovery Analysis 65
3.10 Feature of Gridsim Resource Failure Modified 68
3.11 System Model 69
3.12 Application Model 70
3.13 Performance Evaluation Criteria 70
CHAPTER FOUR: ANALYSIS OF RESULTS 74
4.1 Introduction 74
4.2 Results 74
CHAPTER FIVE: SUMMARY, CONCLUSION AND RECOMMENDATION 83
5.1 Summary 83
5.2 Conclusion 83
5.3 Recommendation 84
REFERENCES 85

LIST OF FIGURES

Figure 1.1: Grid Computing Environment 2
Figure 2. 1: Different categories of Grid Systems 14
Figure 2. 2: Common Grid Scheduler Architecture 19
Figure 2. 3: An illustration of how nature ants lead to identify the shortest path 22
Figure 2. 4: Basic Architecture of fault tolerance in grid computing 32
Figure 2. 5 : Moallem's Proposed Scheduling Pseudocode 43
Figure 2. 6 : Moallem's Scheduling Algorithm 43
Figure 3. 1: An illustration of the ACO Model 56
Figure 3. 2 : Next Node Decision Strategy 58
Figure 3. 3: LookUp Table Algorithm 58
Figure 3. 4: Proposed Scheduling Algorithm 59
Figure 3. 6: Architecture of the proposed fault tolerance in grid computing 62
Figure 3. 7 : Algorithm of the Fault Index 63
Figure 3. 8: Checkpoint Algorithm 63
Figure 3. 9: An illustration of the recovery analysis 67
Figure 4. 1: Graph of average Makespan for varied Gridlets 77
Figure 4. 2: Graph of average Throughput for varied Gridlets 78
Figure 4. 3: Graph of Average Turnaround time for varied Number of Gridlets 79
Figure 4. 4: Graph of Average Makespan time for varied Number of Resources 80
Figure 4. 5: Graph of Average Throughput time for varied Number of resources 81
Figure 4. 6: Graph of Average Turnaround time for varied Number of resources 82

LIST OF TABLES

Table 3. 1: Index Table 57
Table 3. 2: Grid Resource Characteristics 69
Table 3. 3: Gridlet Characteristics 70
Figure 4. 1: Graph of average Makespan for varied Gridlets 77
Figure 4. 2: Graph of average Throughput for varied Gridlets 78
Figure 4. 3: Graph of Average Turnaround time for varied Number of Gridlets 79
Figure 4. 4: Graph of Average Makespan time for varied Number of Resources 80
Figure 4. 5: Graph of Average Throughput time for varied Number of resources 81
Figure 4. 6: Graph of Average Turnaround time for varied Number of resources 82

CHAPTER ONE: INTRODUCTION

1.1 Background of the Study

Computational approaches to problem solving have proven their worth in almost every field of human endeavor. Scientists in fields such as health, meteorology, astrophysics and many more are in need of huge processing power to perform complex calculations in a reasonable amount of time. It might take decade to run a set of modeling experiments on a standard personal computer. Buying a supercomputer costs millions of dollars and thousands more each year to maintain it. That's not to mention the hefty electric bill to keep the massive system running. The standard Personal Computers today have great processing power. The standard tasks for an average user's computer vary very little, usually including word processing, Internet browsing, spreadsheets and presentations.

Owning to the fact that high performance computing resources are expensive and hard to access, option was to use confederated resources that could comprise computation, storage and network resources from multiple geographically distributed institutions (Foster *et al.*, 2008). As most systems are idle for significant periods of time, it should be possible to harness their idleness or unused resources and apply them towards projects in need of such resources. The Grid paradigm now emerged, led by Foster, Carl Kesselman, and Steve Tuecke (Foster *et al*, 2008; Rhodes, 2006) called "fathers of Grid" (Haque *et al*., 2012). They got together to develop a toolkit to handle computation management, data movement, storage management and other infrastructure that could handle large grids without restricting themselves to specific hardware and requirement(Barboni, 2011).

Grid emerges from solving computational problems which otherwise cannot be solved by single personal computer. Such computational problems are financial modeling, weather modeling, data visualization etc. This extremely high computing power is achieved by optimal utilization of distributed heterogeneous resources which are lying idle. This has enable scientists to broaden their simulations and experiments to take into account more parameters (like large values) than ever before. Imagine millions of computers owned by individuals and institutions from various countries across the world connected to form a single, huge, super-computer so as to utilize the resources as depicted in Figure 1.1



Figure 1.1: Grid Computing Environment

(http://eu-datagrid.web.cern.ch/eu-datagrid/images/images/grid-small-prov.jpg)

The term Grid is analogous with "electrical power grids", that provide consistent, pervasive, reliable, transparent access to utility power irrespective of location source(Rhodes, 2006; Pritpal and Gurinderpal, 2013; Foster and Kesselman, 2002).

Grid computing is concerned with coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations. The key concept is the ability to negotiate resource-sharing arrangements among a set of participating parties (providers and consumers) and then to use the resulting resource pool for some purpose (Foster, 2002).

Grid enables the sharing, selection, and aggregation of a wide variety of geographically distributed resources including supercomputers, storage systems, data sources, and specialized devices owned by different organizations for solving large-scale resource intensive problems in science, engineering, and commerce(Buyya, 2002).

1.1.1 General Issues in Grid System: Principles

Four main aspects characterize a grid(Rhodes, 2006; Buyya, 2002):

1. Multiple Administrative Domains and Autonomy:Grid resources are geographically distributed across multiple administrative domains, most often in different time zones, owned by different organizations. It is obliged to honor the autonomy of resource owners together with their local resource management and usage policies.
2. Heterogeneity: A Grid involves a collection of resources, heterogeneous in nature and will comprise enormous range of technologies.
3. Scalability. A Grid might grow from a small number of integrated resources to millions. This led to problem of potential performance degradation as the size of Grids increases. Subsequently,applications requiring numerous geographically located resources must be designed to be latency and bandwidth tolerant.
4. Dynamicity or adaptability: With so many resources in a Grid, the probability of some resource failing is high. Therefore resource failure should be considered as a rule rather than the exception. Applications or Resource managers must adapt their behavior dynamically and use the available resources and services efficiently and effectively.

Due to dynamic natureof grid, more failures are likely to occur in grid environments, thereby affecting the time needed to execute job applications and therefore degrading the performance of the system. Grid compute intensive applications often require much longer execution time in order to solve a single problem. The huge computing potential of grids, usually, remains unexploited due to their susceptibility to failures like, process failures, machine crashes, and network failures etc (Garg and Kumar, 2011). The failure of a resource running a user job has a huge effect on the Grid performance. Hence, in order to ensure the high system availability, the job site failure handling is inevitable.In grid computational system, incorporating fault tolerant algorithms in grid scheduling is advocated. 'Swarm Intelligence for Distributed Job Scheduling on the grid algorithm' proposed byMoallem (2009) will be enhancedby incorporating Fault tolerant technique.

1.2 Problem Statement

Grid environment are dynamic in nature; where resources are geographically distributed, heterogeneous in nature, owned by different individual or organizations with their own policies, having different access models, and have dynamically varying loads and availability.In Grid scheduling system,the availability and performance of grid resources are unpredictable as requests from within an administrative domain may gain more priority over requests from outsideand alsonew resources may join or become unavailable, this results in performance fluctuation, which makes it difficult to implement(Caminero *et al.*,2007) the scheduling methods of parallel computing in Grid environment. These Resources are not dedicated to grid application due to site autonomy and competition by applications for resources and also the capabilities of the computing resources varies, and jobs often arrive dynamically. As the number of components in a distributed system

increases, the component failure rate also increases (Townend and Xu,2003).Most errors likely to occur in grid environment may be intensified by the fact that many Grid applications will perform long tasks that may require several days of computation, if not more. In addition the cost and difficulty of containing and recovering from faults in Grid applications is higher than that of normal applications (Gupta, 2011). While some Grid Scheduler in computational grid, assume resources are always available and 100% reliable and there is no failure of resourceswhile processing group of tasks.Failure and unavailability of resources are inevitable on real Grid environments and certainly will affect users Quality of Service (QoS) requirement. Working with such assumptions will lead to the simulated model outperforming the real grid system. Most existing fault management mechanisms reacts when failure has occurred and application dependent. For example if a job execution machine fails during execution; jobs would be submitted on another machine from start. And this is too expensive to afford for computational intensive jobs that require huge computational resources (Arora *et al*, 2013).To achieve the promising potentials of computational grids, an effective and efficient scheduling system is fundamentally important.Research'sare been carried out on incorporating fault tolerance in computational grid environment where some researchers have implemented fault tolerance in most scheduling algorithms. This research incorporated fault tolerance in Ant Colony Optimization (ACO) scheduling algorithms carried out in Moallem (2009).

1.3 Research Question

This research work attempts to answer these questions:

1. What scheduling mechanism and programming tools will be used for efficiency?

2. What fault technique will be employed to solve the problem discussed in the problem statement?
3. How efficient is the technique used?
4. How willthe proposed technique be validated to prove its efficiency?

1.4 Research Motivation

Most of the scheduling algorithms assumed resources are fully reliable and there is no failure of resource while processing group of tasks. Grid environment involves multiple resources which are heterogeneous in nature and geographically distributed. These Resources are computing system, specialized devices, distributed applications etc. the probability of some resources failing is high. If the grid resource failed during execution of the job, the job is rescheduled on another resource which starts executing the job from scratch. Some of these resources satisfy the user resource criterion (like processors speed, computational cost,storage capacity etc) but they have a tendency toward faults. In such a scenario, the scheduler goes ahead to select the same resource for the simple reason that the grid resource assured to meet user's requirements of the jobs. This leads to more time consumed for the user job than expected. This is a setback to the Grid providers and users.Researchers are making effort to see that fault tolerant scheduling algorithms are incorporated into scheduling algorithms in grid environment.

1.5 Research Aim and Objectives

The aim of this dissertation is to Improve Job Scheduling Algorithm in Grid Computing Environment Using Fault Tolerance Mechanims. Therefore, the research objectives include proposal of a fault tolerant scheduling algorithm that will:

1. Enable job execution in spite of resource failure in the context of grid computing environment.
2. Improve user Quality of Service (QoS) requirements (e.g. deadline to complete its execution).
3. Reduce the selection probability of resources with more fault occurrence history.

1.6 Research Methodology

The following are the methods that were adopted for this research work:

1. Incorporatethe fault tolerant mechanismintoscheduling algorithms proposed in Moallem (2009).
2. Implementation and simulation of the proposed scheduling algorithms using GridSim Toolkit-5.2.
3. Evaluate the performance of the proposed scheduling algorithm with Moallem (2009).

1.7 Scope and Limitation

The scope of this research work is to incorporate fault tolerance mechanism; checkpoint technique into scheduling algorithm proposed by Moallem (2009). Checkpoint strategy saves the process state at a particular interval of time of each job to be executed on the resource. Checkpoint strategy provides an efficient way to guarantee the completion of jobs according to the QoS required by the user.

Limitation of the research work, assumed that the storage device is dedicated and reliable. And also that resource does not fail when and during checkpointing time and also when recovery is being done.

1.8 Contribution to Knowledge

The following are the contribution made:

1. The need for a job scheduling strategy with fault tolerant mechanism for a computational grid environment is advocated. Architecture for in-cooperating fault tolerant based job scheduling model inMoallem (2009) dissertation is proposed. This will enable the jobs to start from the last checkpoint after resources failure not from scratch as in Moallem's.
2. The proposed scheduling strategy uses job checkpointing recovery mechanism to checkpoint the application state at regular intervals so that jobs can restart from the last checkpoints upon resource failure, this enables the grid to complete the jobs within short period of time.
3. The proposed scheduling algorithm consider not just the load information of the resources as in Moallem's but also the fault rate of the resource when selecting an optimum resource for executing the job.
4. The proposed fault tolerance scheduling is implemented using GridSim Toolkit simulator, a java-based discrete-event Grid simulation toolkit. The GridSim toolkit is one of the most widely used grid simulation tools.
5. Performance evaluations were carried out by comparing the proposed scheduling algorithm with Moallem (2009) which is without fault tolerant mechanism. The proposed algorithm improvesthe user job execution time and system throughput.

1.9 Dissertation Organization

The organization of the rest of the dissertation and a brief outline of the chapters in this thesis is as follows. In chapter 2, a review of related scheduling algorithms and fault

tolerance mechanisms in grid environment is carried out. In chapter 3,the gridsim simulator and the methodology used in the proposed swarm intelligence job scheduling with fault tolerant, an enhancement of Moallem (2009) with details of scheduling model use and the architecture of the scheduling algorithm is discussed. In Chapter 4, involves the design and implementation of a prototype system and performance evaluation of the proposed system with Moallem (2009). Chapter 5 concludes the thesis by summarizing the research thesis and outlining future work.

1.10 Definition of Terms

ACO: Ant Colony Optimization; simulate the behavior of ant to find desirable solutions to difficult combinatorial optimization problems.

Checkpoint latency: The duration of time required to save the checkpoint.

Checkpoint overhead: The increase in the execution time of an applications caused by the check pointing.

Computing resources: resources which allow users to execute the required application.

Error: is the part of the system's state that may lead to a failure.

Failure: is an event that occurs when the delivered service deviates from its correct behavior. A service is said to have failed if it could not comply with the promised functional specifications.

Fault: is the adjudged or hypothesized cause of an error. A fault is an unwanted but possible constellation of system states.

Gridlet: is a package that contains all the information related to the job and its execution management details such as job length expressed in MIPS, disk I/O operations, the size of input and output length, and the job originator.

Grid Information System (GIS): contains information about the available grid resources with their computing capacity and cost at which they offer services to grid users.

Grid Resource Broker (GRB): maps jobs received from users to the grid resources.

Grid Scheduler: schedule jobs to grid resource.

Grid scheduling: process of making scheduling decisions involving resources over multiple administrative domains.

Global scheduling is the problem of deciding where to execute the job. Global scheduling is not responsible for the final mapping of the job's processes onto CPUs, this done by the machine's local scheduler, i.e., operating system.

Heterogeneity: Resources are said to be heterogeneous when they do not have all the exact same characteristics. For instance, the set of processors in a grid is heterogeneous if they do not all have the same processing capabilities or the same amount of memory.

Heuristics: refers to experience-based techniques for problem solving, learning, and discovery.

Job is the user's program or application which is executed in the Grid. The job is also subject of the scheduling process.

Job Migration: Sometimes it happens that due to some reason a job cannot be completely executed on a particular machine. At the time of failure of any task, task can be migrated to another machine.

Local scheduling is used to assign the process (es) of the job to the CPU(s) of the single machine.

Makespan: is the time at which the application execution is completed.It is measured from the time the first job is sent to the grid, until the last job comes out of the grid.

Meta-heuristic is a computational method that optimizes a problem by iteratively trying to improve a candidate solution.

Proactive faulttolerant: The failure consideration for the grid is made before the scheduling of a job and is dispatched with hopes that the job does not fail.

Process: referred as a task, is an instance of a program in execution.

Post-active: A mechanism handles the job failures after it has occurred.

Quality of Service (QoS): is the ability to provide different priority to different jobs and users, or to guarantee a certain level of performance to a job.

Re-execution: A task whose execution is not reliable enough (for instance, because its execution speed is below a threshold) is executed twice (or more) to enhance its reliability.

Resource: any hardware or software component such as a cluster, a supercomputer or a storage repository. Resources perform an operation, such as memory for data storage, processor for data processing, bandwidth for data transfer and so on.

Resource Characteristics: object that stores the properties of a Grid resource: architecture, OS, list of Machines, allocation policy: time- or space-shared, time zone.

Resource Scheduling: the resource scheduling is the process of mapping resources based on the requirements, characteristics which are specified in a query.

Resource selection: is the problem of deciding which, among all available resources, should be used to solve a given problem.

Schedulers: are responsible for scheduling, and it manages various tasks – resource discovery and resource selection - required for effective application execution.

Scheduling: is the problem of deciding at what time (when), and on which resource (where), to execute each of the (atomic) tasks that must be executed on the given platform.

Site: Sites are autonomous unit composed of one or multiple resources. It is also termed as node.

System-centric: is a traditional scheduling approach which is commonly applied in single administrative domains by attempts to optimize system-wide measures of performance. They aim to enhance the system throughput and utilization, and to thereby complete execution at the earliest possible time.

Task: is the smallest unit that can be scheduled by the scheduler and allotted to a resource. Task has parameters such as priority, memory requirement, deadline, etc.

Task scheduling: Scheduling refers to assigning of tasks to a group of resources that are distributed in multiple domains.

Task Resubmission: A job may fail now whenever a failed task is detected, In this case at runtime the task is resubmitted either to the same or to a different resource for execution.

Throughput:number of processes completed per unit time

Turnaround Time: mean time from submission to completion of process

User-centric: concentrate on users' requirements by delivering maximum utility to the users of the system based on their QoS requirements. For example, a guarantee of certain QoS based on the attributes that the user finds important, such as the deadline by which the job has to be completed.

CHAPTER TWO: LITERATURE REVIEW

2.1 Introduction

The scheduling problem is not just limited to Grid systems. In fact, it is one of the most studied problems in the operation and optimization research communities. However, in the case of Grid systems, the scheduling problem is different and more challenging (Abdullah, 2011). Also job scheduling on Parallel Computers cannot be applied in grids environment in that they have high-bandwidth, low latency communications links between them, and are all under the central control of a single authority (Kelly, 2004), scheduling in this context is a simpler problem because the scale of the machine is limited, and the scheduler has complete control over every program that is executed. This is significantly different from a grid environment, where additional work may be performed by resources at the discretion of individual owners. Various research works have been done on job scheduling problem in grid, but still further analysis and research need to be done to improve the performance of scheduling algorithm in computational grid.There exist many scheduling algorithms, and considering all of them will be a very long and tedious process; therefore, this research work will consider and exploit some current scheduling algorithms related to this research work.

2.2 The Grid System

2.2.1 Types of grid system

Different types of grid architecture are designed to address different types of problems and computing needs. Some grids are designed to take advantage of improved processing resources, whereas some are designed to support collaboration among different resources. Grid architectures can be grouped into three categories according to the functionalities they

perform: as depicted in Figure 2.1(Rhodes, 2006; Yadav*et al.*, 2013) classify grid systems into three categories:

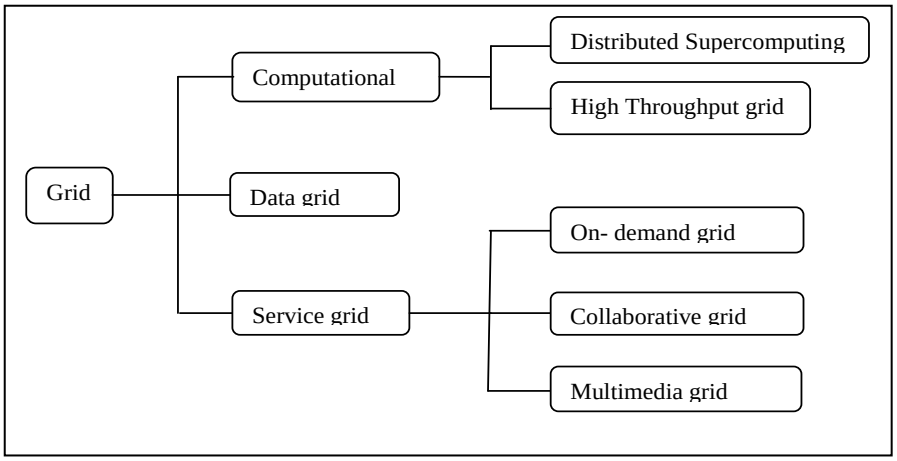


Figure 2. 1: Different categories of Grid Systems (Yadav*et al.*, 2013)

1. **Computational grid**: is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities(Rhodes, 2006; Yadav*et al.*,2013).It provides access to large pool of shared processing power suitable for high throughput applications and computational intensive. Computational grid is further splited into Distributed Supercomputing grid and High Throughput grid depending on how resources are utilized(Yadav *et al.*, 2013). A supercomputing grid executes applications on multiple machines in parallel, minimizing the execution time of a task. While A high throughput Grid aims to utilise idle computational cycles to process a stream of different tasks.
2. **Data grid :**is the storage component of a grid environment. Scientific and engineering applications require access to large amounts of data and often this data is widely distributed. Users are not concerned with where this data is located as long as they have access to the data.

3. **Service grids**: provide applications for the proper functioning of the grid e.g. maintenance services, resource analysis, task scheduling, security etc (Abdul Azeez and Haque, 2012). A service grid provides services that cannot be accomplished by any single local machine. The service grid is further categorized into: on-demand grid, collaborative grid and multimedia grid systems(Yadav *et al*., 2013; Abdullah, 2011).

2.2.2 Application of grid computing

The Grid infrastructure is bringing together distributed resources such as supercomputers, data sources and specialized measurement systems to enable collaborative working on interdisciplinary projects that could not be otherwise possible. Grid infrastructure has benefit in many applications in science and industry. Systems in grid, try to utilize resources which are idle.This is aimed at avoiding wastage, thus enable Scientist, commercial and other users to carry out theirapplications (which may include compute intensive enormous applications data like nuclear financial modeling, protein folding, climate\weather modeling, image rendering, earthquake engineering etc) (Dabas and Arya, 2013),thus, solving complex problem to the computational level of a supercomputer.

Major benefits which can be utilized by application of grid are as follows:(Jacob *et al.*,2005; Dabas and Arya, 2013):

a. Improved efficiency/reduced costs.
b. Optimized utilization of underutilized resources.
c. Exploiting underutilized resources
d. Virtual resources and Virtual Organization (VO)
e. Increase capacity and productivity
f. Parallel processing capacity

g. Resource balancing

h. Heterogeneous system support

i. Reduced time of result.

The areas in which grid computing can be applied are as in: (Dabas and Arya, 2013)

1. Engineering Design and Automation: Computational aerodynamics, artificial intelligence and automation, finite-element analyses, remote sensing applications, pattern recognition, computer vision, image processing, etc.
2. Medical, Military and Basic Research: Polymer chemistry, medical imaging, nuclear weapon design, problem of quantum mechanics, etc.
3. Predictive Modeling and Simulation: Flood warning, socio-economic and government use, numerical weather forecasting, astrophysics (Modeling of Black holes and Astronomical formations), semiconductor simulation, Oceanography, human genome sequencing, etc.
4. Energy Resource Exploration: Plasma fusion power, seismic exploration, nuclear reactor safety, reservoir modeling, etc.
5. Visualization: Computer-generated graphics, films and animations, data visualization, etc.

2.2.3 Challenges to grid computing

Distributing systems are responsible for providing the main execution platform for High Performance Computing (HPC). These systems can be homogeneous(such as cluster) or heterogeneous (such as grid and cloud etc).HPCsystems are prone to different kinds of

problems (Haider*et al.*, 2011). In Grid environment,challenges facedare: Security, Grid reliability, Resource Selection, Quality of Service, Fault Tolerance and Policy Heterogeneity. Further detailsarefound in (Dabas and Arya, 2013).With the above challenges listed, the focus is on resource selection with fault tolerance in a Grid environment.

2.3 Resource Management System

Resource management in grid computing areoperations that control the way grid resources and services are made available for use by entities like users, applications and services (Khanli and Analoui, 2008).This is to ensure efficient utilization of computer resources and optimization performance of specific tasks. Resource management in computational grids, deal with identifying requirements, matching resources to the applications, allocation of resources, scheduling and monitoring of the grid, in a timely manner(Singh, 2009).

Resource management systems in orthodox system cannot implement effective mechanisms and policies for the efficient use of the resources. They are designed to work under the assumption that they have full control of a resource, whereas, such assumption cannot be applied to the Grid. When dealing with the Grid, methods for managing Grid resources across separate administration domains must be develop. Hence the grid resource management system has to take care of resource heterogeneity, loss of absolute control, and inevitable differences in scheduling policies and security mechanisms(Moallem, 2009; Mukherjee, 2011; Ekmecic and Milutinovic, 1996; Quinn, 1994; Crichlow, 2003; Foster and Kesselman, 1999). Grid resources include processor clusters, supercomputers, storage devices, and related hardware, together with operating system and other software for managing these resources. Factors such as security, fault tolerance along with scheduling

make the process of resource management far more challenging (Foster and Kesselman, 1999; Ekemecic and Milutinovic, 1996).

2.4 Grid Resource Scheduling

Scheduling in general refers to the allocation of resources over a period of time for specific purposes (Kelly, 2004). Decisions on which resources are use and when, are made based on information about the resources themselves, and the ways in which applications wish to utilize them. The primary goal when carrying out scheduling is to optimize for one or more variables, such as job resource utilization or completion time. Scheduling application on grid resources is one of the challenges faced in grid resource management. It is a process that maps and manages the execution of inter-dependent tasks on the distributed resources (Suryadevera *et al.,* 2012). A Grid Scheduler is different from local scheduler in that a local scheduler only manages a single site or cluster and usually owns the resource, whereas Grid Scheduler is in charge of resource discovery, Grid scheduling (resource allocation and task scheduling), and job execution management over multiple administrative domains(Singh, 2009). As depicted in Figure 2.2, Local resources lie at the bottom of the architecture each being managed by a local resource manager (LRM).

LRM enables low level processing of requests and executionof job that satisfiesa request, remote monitoring management of the jobs as well as updating the information service with the current availability and capabilities of the resources that it manages.

Information Service Component: maintains the various resource characteristics and answers to the queries for acquiring the information about the resources. A sufficient scheduler must incorporate the current information of resources when making scheduling decisions.

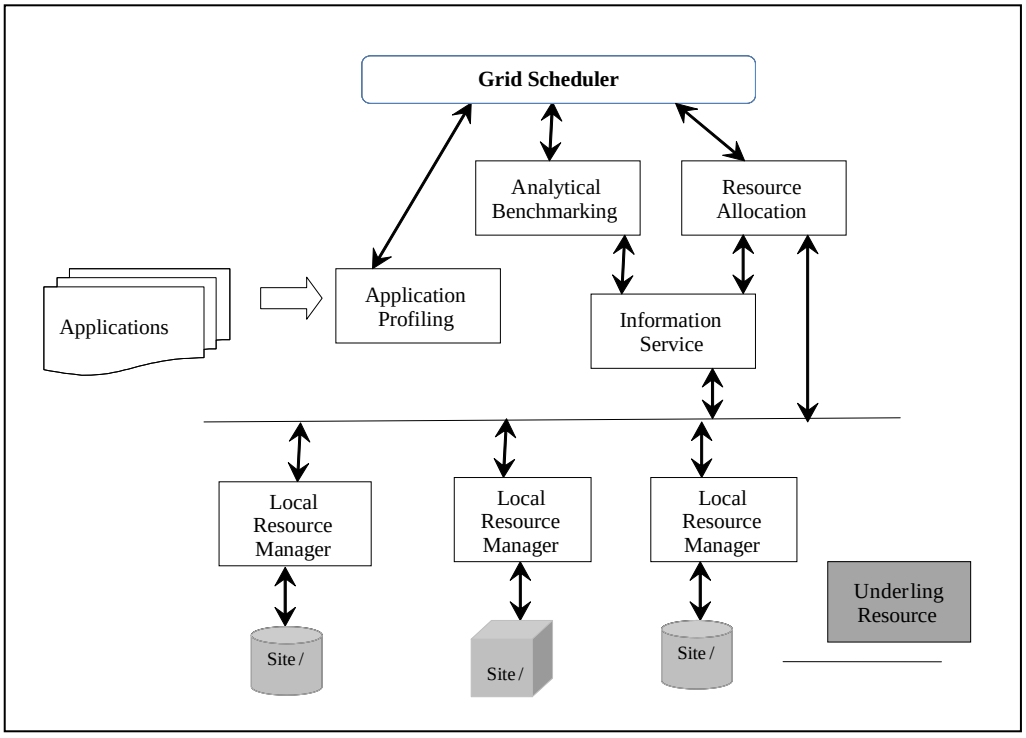


Figure 2. 2: Common Grid Scheduler Architecture**(**Sigh**,** 2009**)**

Application profiling interface provides a way for user to submit jobs to the grid where as the analytical benchmarking component provides a measure of how well a computational resource will perform on a given job. The Grid scheduler (also known as resource broker) is the backbone of this architecture. Its major responsibilities are resource selection(which is the process of selecting feasible and available resources for a given application) and also mapping (which is the process of placing the jobs on to resources for execution). Scheduling is aimedat providing amongst others - two objectives namely(Rajeshkumar and Kowsigan, 2011):

a. High performance computing : decreases the execution time

b. High throughput computing: increases the processing capacity of the system.

There are three main phases of scheduling on grid resources (Schopf, 2002); First phase;Resource Discovery where all the available resource will be generated. Second phase; Resource Scheduling, gathers the information about the resources and chooses the best suitable resource to match the application requirements. Third Phase, executes the job in the selected machine which include file staging and cleanup. Resource scheduling is an NP-Hard problem (Fidanova and Durchova, 2006)and Grid Resources are unpredictable as organization can decide whether to leave or join a Virtual Organization at any time, so conventional scheduling will be unrealistic in this system.

It is essential to assign appropriate resources to tasks which can only be done by a good scheduling means, so that the system can perform better and applications can avoid unnecessary delays.

2.5 Artificial Intelligence (AI)

Artificial Intelligence is used to describe research into human-made systems that possess some of the essential properties of life (Sarangi, 2009). AI studies how biological techniques can help out with computational problems. There are lots of computational techniques inspired by biological systems. Example of AI are as follow; artificial neural network is a simplified model of human brain; Genetic algorithm is inspired by the human evolution, swarm intelligence, more specifically, the collective behaviors of simple individuals interacting with their environment and each other (Sarangi, 2009).

2.5.1 Using swarm intelligence techniques

The concept of swarm intelligence stems from observing the behaviors of social groups of organisms, such as ants, bees, geese, fish and other species. These kinds of intelligent

patterns need relative number of intelligent individuals to achieve certain types of problem solving capabilities (Wu, 2014). The benefit of these techniques is thecapability to search large search spaces, which arise in many combinatorial optimization problems, very efficiently (Subrata and Zomaya, 2003). Swarm intelligence research has two main algorithms (Moallem, 2009; Wu,2014): Particle Swarm Optimization (PSO) and Ant Colony Algorithm (ACO)

1. Particle swarm optimization (PSO) is a population based stochastic optimization technique developed by Dr. Ebehart and Dr. Kennedy in 1995, inspired by social behavior of bird flocking or fish schooling (Sarangi, 2009). PSO algorithm is not only a tool for optimization, but also a tool for representing socio cognition of human and artificial agents, based on principles of social psychology. PSO as an optimization tool provides a population-based search procedure in which individuals called particles change their position (state) with time.
2. Ant Colony Optimization was inspired by the behaviors of ants and has many successful applications in discrete optimization problems (Sarangi, 2009). Ant when compared to humans has a very little brainpower. While humans have approximately 10 billion neurons, ant has only 250,000. The real power of ants resides in their colony brain. The self-organization of those individuals is very similar to the organization found in brain-like structures. Like neurons, ants use mainly chemical substances to communicate; one ant releases a molecule of pheromone that will influence the behavior of other ants. As ants traverse a path, they deposit pheromones. Pheromones are chemical substances that attract other ants and are deposited by ants as they travel.

Hence, they choose the way with higher probability paths, which are marked by stronger pheromone concentrations, as depicted in Figure 2.3.

ACO utilizes search experiences represented by pheromone and domain knowledge represented by heuristic information to accelerate the search process (Parvatkar and Gore, 2014).

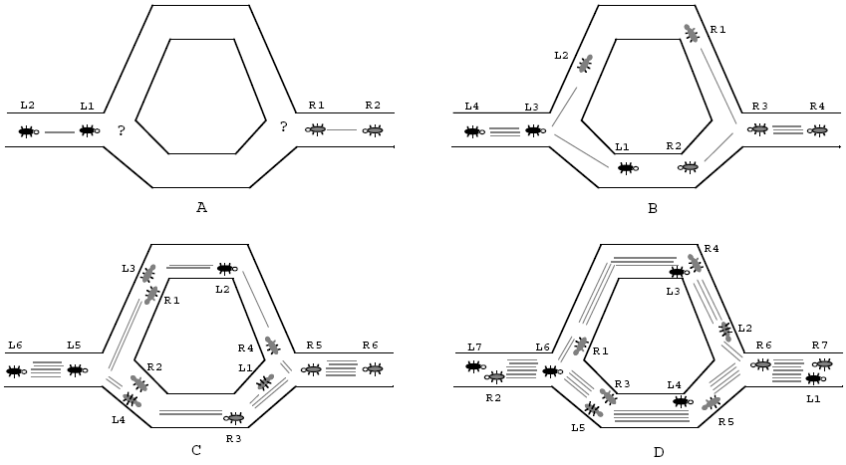


Figure 2. 3: An illustration of how nature ants lead to identify the shortest path (Lorpunmanee *et al.*, 2007)

The basic Pseudo code for ant colony optimization (ACO) is given below (Sasikala and Ramesh, 2014):

1. Initialize parameters
2. Initialize pheromone trails
3. Create Ants
4. While stopping criteria is not reached do
5. Let all ants construct their solutions to make the resultant solution
6. Update pheromone trails
7. Allow the Processes to execute
   End while.

2.6 Fault Tolerance

Fault tolerance is responsible for handling the reliability and availability of distributed systems (Haider *et al.,* 2011). Fault tolerance is a capability developed in the system so that it

could perform its function correctly even in the presence of faults.Also, when fault occurs it is developed to detect it immediately and recovers the executable task without participation of any external agents, thereby,making the system more dependable. In fault tolerance, According to Selic (2004), Garg and Kumar (2011) failure is encountered when a system moves away from its particular behavior. The cause of a failure is called error, which also ultimately depicts some sort of fault or defect in that system. That is, fault is the actual cause of a failure, and error is just an indication or sign of a fault. Multiple errors could be due to a fault, and even a single error could be the cause of multiple failures. With many independent resourcescooperating together as one, the chance of failure of an individual resource increases dramatically, particularly if the resources are very physically dispersed and connected using the internet (known for its unstable medium). With the possibility of many thousands of computing resourcesoperating together, the odd of a long running process not failing on at least one resource is almost zero (Orbell, 2002). Also, according toTownend and Xu, (2003) that as applications scale to take advantage of Grid resources, their size and complexity will increase dramatically. Those systems with complex asynchronous and interacting activities are very prone to errors and failures due to their extreme complexity.

Thereforefailure free applications are unfeasible irrespective of the fault avoidance and fault removal techniquesimplemented (Townend and Xu, 2003). It is morelikely errors will beaggravated by the fact that many Grid applications will perform long tasks that may require several days of computation, if not more.

Fault detection in Grid resources are two strategies: Pull and Push model as described in (Li and Lan, 2006). In the pull model, different grid components are responsible for sending periodic signals to a fault detector, announcing that they are alive. In the absence of any such

signal from any grid component, the fault detector recognizes that failure has occurred at that grid component. It then implements appropriate measures dictated by the predefined fault tolerance mechanism. In the push model, it is the responsibility of the fault detection component to send periodic signals (live requestsmessages; For example, “Are you alive?”) to the different grid components (Li and Lan, 2006). Handling of failures by scheduling strategies can occur either before or after the scheduling of the resources(Amoon, 2011). The first approach of handling failures is called Pro-active and the second is called Post-active. Post-active approaches, also known as techniques of job monitoring, are relatively easier to implement. On the other hand, pro-active approaches require more information about grid resources and works in a probabilistic fashion. There are three mechanisms use to deal with fault-tolerant issues in grids*(*Amoon, 2012).The first category is called Space Redundancy or Job Replication. In this category, the same job is replicated to be executed on multiple undependable resources to guard the job against a single point of failure.The replication avoids task re-computation by executing several copies of the same task on more than one compute stations. The second category is called Time Redundancy or Checkpointing and Rollback. In this category, the state of a running job is saved to a stable storage. When a resource failure occurs, the job resumes execution from the last process state. More on checkpoints will be discussed later in this chapter. The third category is called Adaptive. It combines job replication with checkpointing to achieve the fault-tolerant. Here, only one replica will be executing and all other replicas are updated periodically, so that if active replica is not available during the recovery of a resource failure the other replicas will be used (Srikala and Ramachandram, 2013).

Moreover, the heterogeneous nature of Grid resources means that many Grid applications will be executing in environments where interaction faults are more likely to occur between disparate Grid resources, whilst the dynamic nature of the Grid; resources may enter and leave at any time, in many cases outside of the applications control.Therefore a Grid application must be able to tolerate (and indeed, expect) resource availability to be unstable. It is therefore necessary to explore and incorporate the implementation of fault tolerant techniques for Grid scheduling algorithms as being proposed in this dissertation work.

2.6.1 Checkpointing

Checkpointing is one of the prevalent techniques used to provide fault-tolerance on unreliable systems (Garg and Kumar, 2011). Checkpointing is being applied to transaction processing systems as well as to batch programs. It is being applied in production environments, real time systems (Wolter, 2007) scientific computing, mobile computing, distributed database, telecommunication and critical applications in distributed networks (Patial and Thakur, 2014). With checkpointing the system or program state is saved to permanent storage and if a failure happens after the checkpoint, the system or program does not need to restart all over from the beginning but only from the most recent checkpoint. Taking a checkpoint often is called a save, while restarting from the most recent checkpoint is called a rollback. Checkpoint can be stored on temporary as well as stable storage. When checkpointing a process, the rollback will restores an earlier state of the process.

A. Classification by Checkpointing Protocol

Checkpointing protocol defines how to make global snapshot of a distributed application (Batrashev, 2007). This approach studies methods and algorithms to synchronize local

states of processes in a distributed application (Batrashev, 2007). It solves the problem of combining checkpoints of separate processes and communication channels into a meaningful global checkpoint. The solution for this problem is not trivial because there is no shared clock to take snapshots simultaneously. This technique is divided into checkpoint based and log-based.

1. Checkpoint-based

Checkpoint-based rollback recovery relies only on checkpoints of processes and channels and does not use assumptions about piecewise deterministic (PWD) restores the system state to the most recent consistent set of checkpoints whenever a failure occurs (Patial and Thakur, 2014).The different kinds are as follows: full checkpoint, incremental checkpoint, uncoordinated checkpoint, coordinated checkpointing etc

2. Log-based

Log-based rollback recovery or message logging protocols are based on piecewise deterministic (PWD) assumption (Hellmuller, 2011). It assumes that a program has periods of deterministic execution which are separated by non-deterministic events. These non-deterministic events are recognized and can be logged in a determinant containing all non-deterministic information. Based on checkpoints and logged determinants, a message logging protocol is able to recover the exact state before the occurrence of a failure. The different kinds are pessimistic logging, optimistic logging and casual logging.

B. Classification by Checkpointing Mechanism

It is not obvious which data should be considered as process state and it is usually dictated by the requirements to the system under consideration. For example, should the state of the

libraries be considered as part of the state of the process or whether the ID number of the process must be preserved? It can be accomplished on many levels: by specific hardware, operating system, linked library or by application itself. Traditionally, optimizations of the checkpointing process are targeted toward scientific programs written in Fortran or C. Such programs often have good locality and large regions of read-only data (Lawall and Muller, 1999). Each checkpoint contains only the pages that have been modified since the previous checkpoint. Two main approaches exist; System- and Application- level checkpoining.

1. **System –level checkpointing (SLC)** saves the states of the whole sections of the application without knowledge of internal data structures or algorithms. System-level checkpointing operates on the level of system library or operating system itself. It uses OS specific mechanisms and knowledge to take a snapshot of the application state. These approaches can be divided into two groups:
    a. User mode - operates on the level of all other libraries and applications and usually does not need any changes in the operating system.
    b. Kernel mode mechanisms - lives in operating system kernel and operates on the level of hardware and software drivers. It does not have any knowledge about application specific behavior and has to save all data in the application. Not all data need to be saved when second checkpoint is taken, only data changed between two sequential checkpoints need to be identified and stored.

2. Application – Level checkpointing (ALC)

Application –level checkpointing (ALC), on the contrary, can use information about the algorithm and can usually change the code of application to add checkpointing

functionality. Application level checkpointing does not use operating system mechanisms to save process state. Special code within the application itself is responsible for this functionality. The code is written by a programmer, generated by some tool or inserted by a compiler. Considering the last variants, there are three models of ALC; Manual, Code preprocessing and Compiler-assisted. ALC has more control over application state, does not have access to the internals of operating system or used libraries and is thus very limited. This approach has two advantages for the developer. First, because it works with the application variables and not with a lower-level representation, the checkpointing code can be made portable without too much difficulty.Furthermore, with a little more effort, it is possible to make the checkpointing data portable as well. Second, the developer is able to write checkpoint code that only saves the minimum set of variables that are needed to restart the application. Programmers implementing ALC into their code must think carefully about what points in the execution are appropriate for checkpointing and which program variables must be saved in order to successfully restart the application. The checkpointing cost can be reduced by reducing the amount of data in each checkpoint. In ALC, this is done by the developer deciding that certain application variables do not need to be saved, because they are not needed or can be recomputed upon restart (Bronevetsky *et al*., 2003). Since SLC systems do not have any knowledge of the application, they rely upon observing low-level details of the application's execution in order to reduce the amount of state saved. Saving application state with ALC is not simple; most programming languages do not give control over global variables and getting a snapshot of the stack is something not present in any widely used language. The reason for such limitations is that conventional algorithms do not require mentioned

features and programming languages, following common patterns, provide means to separate and hide state of different components within application. To clear this point- there is no need to get access to the whole stack state until when implementing checkpointing mechanism.There are two known techniques to save the stack by programming language means. First, the application saves pointers and lengths of local variables every time a function is entered and forgets them upon exiting the function. If the program wants to make a checkpoint it iterates over all saved pointers and saves corresponding memory areas. This approach must ensure that virtual address of the stack is the same upon restart of application. Alternative approach is shown in (Ramkumar and Strumpen, 1997), where the application starts the checkpointing procedure by sequentially returning from function calls and saving values of local variables until it reaches the main function. Then it proceeds in opposite direction by restoring stack state until it reaches start location. During a usual run of the application there is much less steps that must be done and therefore this way is expected to be more efficient. Catching the state of heap variables is no easier than the ones on the stack. Pointers in the C language do not contain information about the type or size of the memory area it points to. Also, there is no heap interface that provides any knowledge or means by which allocated memory areas can be located and extracted. One possible solution is to redefine memory allocation and de-allocation routines and to track this information using additional data structures.

a. Manual ALC: This method of application checkpointing is straightforward and easy to comprehend. The programmer himself is responsible for managing application state and it uses ordinary programming techniques. The programmer decides what data must be saved

and the best location in the program to activate the checkpointing mechanism. The way, he can choose the location where the data amount ids the smallest and there are no in-flight messages in case of a parallel application. The problem here is that it requires new analysis and implementation for every new application and it may be substantial effort for complex algorithms.

b. Code preprocessing method: It analyzes and instruments the code with a special source- to –source compiler. This is called automated Application-level checkpointing (AALC) and is implemented in the C3 (Bronevetsky *et al.*, 2004) precompiler. This approach provides fault tolerance with much less effort than changing the source code manually, but still encounters all issues related to ALC.

c. Compiler-assisted method: A little different approach than using source-to –source compiler is to analyze and instrument the code within the compiler itself.

d. Mixed –level checkpointing: it uses combination of ALC and SLC or even different SLC techniques for different parts of the same system. For example, one may wish to use ALC for his application data, but SLC to store the internal state of the libraries.

Writing checkpointing in Object-oriented programming language, such as Java, place a new challenge on Checkpointing (Lawall and Muller, 1999). Reasons why system-level approach may not be appropriate for java program (Lawall and Muller, 1999):

1. Object-oriented programming style encourages the creation of many small objects.

2. Each object may have some fields that are read-only, and others that are frequently modified. Thus, object encapsulation conflicts with programmer-based data placement strategies.

3. Java programmer has no control over the location of objects, making it impossible to ensure that frequently modified objects are all stored in the same page. Furthermore, a single page can contain both live objects, and objects awaiting garbage collection.

4. Java programs are run on a virtual machine. Some internal state of the virtual machine is not needed to reconstruct the state of the program.

2.7 Checkpoint Interval

Efficiency of a checkpoint mechanism is strongly dependent on the length of the check pointing interval. Checkpointing interval is the duration between two checkpoints. Each interval starts when a checkpoint is established and ends when next checkpoint is established. Frequent checkpointing leads to a large number of redundant checkpoints, which may enhance overhead like delay job processing by consuming computational and network resources. On the other hand, lazy check pointing may lead to loss of significant computation because a substantial amount of work has to be redone in case of a resource failure (Fialho, 2011). Hence, the decision about the size of the checkpoint interval and the checkpoint technique is a complicated task and should be based upon the knowledge about the application as well as the system (Hellmuller, 2011). Most utilized checkpoint mechanisms use resource fault index to determine checkpoint interval. Amoon (2012), shown that resource failure rate is more effective than resource fault index.

2.8 Architecture of Fault Tolerance in Grid Computing

An architectural view of fault tolerant in Grid environment considered in this thesis is depicted in Figure 2.4. The scenario is as follows: The users through a User Interface (UI)

submit their jobs to the grid transparently specifying their QoS requirements such as the cost of computation, deadline to complete the execution, the number of processors, speeds of processing, internal scheduling policy, and time zone. Grid Resource Broker (GRB) which is an important entity of the grid is connected to an instance of a user (Nazir*et al.*, 2009). Each grid job is first submitted to its broker, which then schedules the grid job according to the user's scheduling policy (Nazir*et al.*, 2009).A GRB also called Grid Scheduler (GS) maps jobs received from users to grid resources, scheduling decisions taken by the GS are based upon information provided by the Grid Information Server (GIS), which contains information about all available grid resources with their computing capacity and cost at which they offer their services to grid users. All resources that join and leave the grid are monitored by GIS. The component Fault Handler is responsible for detecting failure of resources and estimating the required information for fault tolerance process. If the fault handler detects a grid resource failure during execution of job, the job is rescheduled on another resource which starts executing the job from scratch. This leads to more time consumed for the job than expected. Thus, the user's QoS requirements are not satisfied.

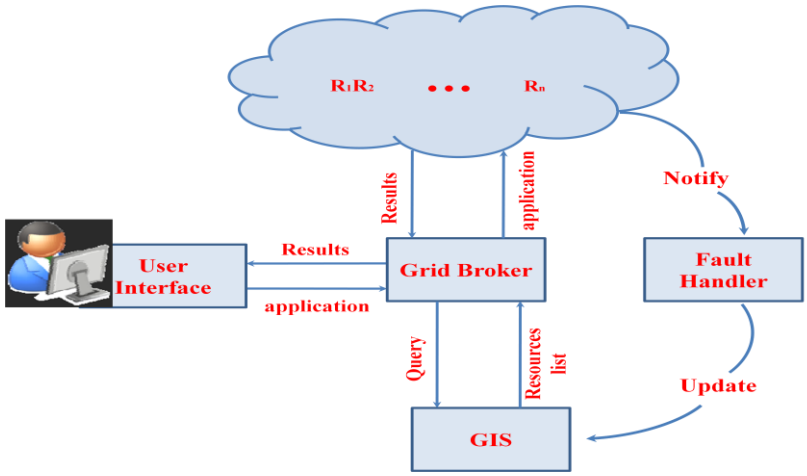


Figure 2. 4: Basic Architecture of fault tolerance in grid computing (Altameem, 2013)

The scheduling system acts not only as an interface between users and grid resources but also provide reliable service to users.

2.9 Review of Some Proposed ACO Scheduling Algorithms

ACO algorithm search may lead to stagnation, when the pheromone trail is significantly higher for one choice than others (Fidanova, 2002). As each ants move from a state i to another state j corresponding to a more complete partial solution.

Fidanova (2002)proposed additional reinforcement of the pheromone of the Ant Colony Optimization algorithms in order to avoid stagnation. This forces the ants to search areas in the search space that has not yet be explored in order to get better solutions. In the beginning, ant constructs feasible solutions and the pheromone trail is then updated. If some movement is not used in current tour, additional pheromone reinforcement is then used.

Yan *et al.,* (2005)proposed that power performance of resource should be chosen with high probability and trail intensity, that is if on path j there is a lot of traffic then it is highly desirable. Introduce a load balancing factor so as to avoid the ant visiting an overloaded resource in order to improve the load balancing capability. Load balancing factor λ of the resource j, is related to the job finishing rate in the resource j to alter the pheromone. The trail intensity changed from

$\Delta\tau_j$ to $\Delta\tau_j + C\lambda_j$ (C>0 is a coefficient of the load balancing factor).

As more jobs finished, the trail intensity increases.Contrarily, the more jobs not completed the decreases in the trail intensity. The proposed algorithms improved the load balance status for the resources in the grid. This method has only address the computation and communication capability of network.

Liu *et al.,* (2006) proposed using ACO for SuperScheduling in computational grid. A superscheduler can submit the jobs to the other neighboring superschedulers when it hasn't the communicational capabilities to run the job or it reaches the maximum resource limit threshold. The Evaporation mechanism introduced modifies pheromone information over time. Pheromone evaporation allows the agents to slowly forget its past history so that it can direct its search towards new direction without being over-constrained by past decisions, preventing the bad solutions to be made by the other agents as the result of dynamicity of the grid resources. The primary aim of pheromone evaporation is to avoid stagnation, the situation in which all the jobs are allocated to same resource.

In Ngenkaew *et al.,* (2007), the authors focused on Ant-Based Clustering algorithm from an ant foraging model's view, and proposed a method of using two kinds of deposited pheromones at each grid, Trailing Pheromone and Foraging Pheromone.

- The Trailing Pheromone (TP): is the pheromone which leads ant to return to the nest. When ant already picked up the item of food, it will follow the maximum amount of Trailing Pheromone to return to the nest to drop the food.
- The Foraging Pheromone (FP): is the pheromone which ant deposits when the food is found. When ant is not carrying any food, it will try to find the minimum amount of Foraging Pheromone around itself. In order to find food that is left for others, ant needs to follow the minimum amount of foraging Pheromone unlike using Trailing Pheromone.

Ant has two states: carrying or not carrying food. When ant goes out from the nest to find food, it will lay TP increasing the level and searches for the minimum amount of FP. Until ant finds the food, FP will be deposited and ant needs to find the maximum amount of TP

to return to the nest. That deposition of pheromones, reduce random walking of the ants to search for food and time taken for clustering of ants to occur.

Lorpunmanee *et al.*, (2007), proposed ACO algorithm considered the requirements of each job, which is independent from other jobs and where only one of job processes in any unit time slot (space sharing). The heuristic desirability of assigning job *j* on machine *i* directly moves to *m*, is inversely proportional to the completion time of job that has been assigned on machine. The initial pheromone is computed using the tardiness time by applying the dispatching rule of minimum completion time (MCT). The aim is to maximize the graph representation, thus the design of the graph is to identify the problem to connect the arcs correspondent for each job on each machine. The algorithm can find an optimal processor for each machine to allocate to a job that minimizes the tardiness time of a job when the job is scheduled in the system.

Mathiyalagan *et al*., (2010), also modified transition probability and pheromone updating rule, in order to allow the ants to share information about good solutions. Allow only the best ant to deposit pheromoneat after each iteration, makes the search much more aggressive and significantly improves the performance of ant colony optimization algorithms. In order to allow ants to forget poor information, each pheromone value is also decayed at this stage, this is implemented with a parameter ρ which takes a value between 0 and 1, if ρ is set to one then no decay will take place, if ρ is zero then each pheromone value will be wiped at each iteration and the pheromone trail is effectively switched off.If the value of α is set to zero, then only heuristic information is used and the ants effectively

perform aprobabilistic search. If value of β is set to zero onlypheromone information is used.

Ali *et al.*, (2010) presented ACO algorithm for load balancing distributed system. Information on resources is dynamically updated at each ant movement. Multiple ant colonies were used such that each node sends a colored colony throughout the network. The ants of the same colored, does not followed the same route, to enable them to be distributed all over the nodes in the system. Each node contains information about other nodes in the system. Upon arrival the ant update the node table. However, the multiple ant colonies do not consider resources capacity and jobs characteristics. This can make matching the jobs with the best resources a difficult task for the scheduling algorithm.

Rajeshkumar and Kowsigan(2011) proposedalgorithm that tends to guarantees efficient resource allocation of the Machines and reduced makespan and execution time. The algorithm uses a new pheromone updating rule and probability matrix calculation formula in order to increase the efficiency of the existing ACO. ACOalgorithm for scheduling, initially started with the value of ET_{ij} (expected time taken) matrix. Number of ants = number of tasks. When a job is submitted to a resource, the expected time taken by machine j to complete task i is first computed. The matrix is NxM entries where N is the number of jobs and M is the number of available resource. Where there is ready time, free time, beginning time of machine.

Rastogi and Nishant (2012) used two types of pheromone; Foraging Pheromone (FP) and Trailing Pheromone (TP). Here the ant would lay down foraging pheromone after

encountering under loaded nodes for searching overloaded nodes. When ant comes up to an under loaded node it will try to find the next path through foraging pheromone. The ants would use TP to find its path to the under loaded node after encountering overloaded node. When ant encounters an overloaded node it will try to trace backward. At initial stage the ants follow the foraging pheromone and update the FP trails. When it comes across an overloaded node it follows the trailing pheromones and concurrently updates the TP trails of the path. After reaching an under loaded node of the same type they update the data structure so as to move a particular amount of data from the overloaded node to under loaded node. Ants then select a random neighbor of this node, and if they encounter an under loaded node they start following the FP to trace an overloaded node. The authors claim an improvement in network performance.

Suryadevera *et al.,* (2012) presented an Ant Colony Optimization algorithm for load balancing in grid computing. Scheduling of task to resource is based on the best resource capability and also balances the load of entire resources on grid. The initial pheromone value is evaluated using the resource parameters like No. of processors (n), MIPS (p), communication bandwidth (S_j) and memory capability (m). The trail intensity at time t is updated. The jobs are allocated to resources based on the pheromone value calculated by considering resource parameter. The pheromone updated by adding encouragement or punishment coefficient depending on the status of the job and load balancing factor to the trail intensity, as a resource gets loaded heavily so as to balance the load. The main objective is to achieve high throughput which then increases the performance in the grid environment.

Kumar and Singh (2012) presented Ant based algorithm to solve the load balancing problem in computational grid called ALBA. The proposed scheduling algorithms assume the pheromone to be associated with resources, rather than path. Assume an initial value of pheromone associated with resources. The increase or decrease of pheromone represents load and depends on task status at resources. Whereby when a task failure is reported on any resource, the resource pheromone will be decrease, reporting that the resource is heavily loaded, task completion increase the pheromone of the resource saying that the resource is lightly loaded. He claimed that the proposed algorithm resulting in an improved utilization of resources.

Barde *et al.*, (2014), aim of this algorithm is to find optimal solution that meets all constraints and optimizes the time and space parameter. The creation of new routes requires the use of a forward ant (FANT) and a backward ant (BANT). A FANT is an agent which establishes the pheromone track to the source node and a BANT establishes the pheromone track to the destination node. A node receiving a Forward ant for the first time creates a record in its routing table. A record in the local routing table is a triplet and consists of (next hop, destination address, pheromone value). When a duplicate of FANTs are encountered through the unique sequence number are destroyed by the nodes. When the FANT reaches the destination system, it is processed in a different way and the node fetches the information of the FANT and destroys it. Subsequently, it creates a BANT and sends it to the source node. The BANT has the same task as the FANT, i.e. establishing a track to this node. When the sender receives the backward ant (BANT) from the

destination, the route is established and data packets can be sent. The forward ant creates only one pheromone track toward the source node, while the backward ant creates two pheromone tracks toward the destination node. It also supports multi-path routing Experiment taken shown enhances performance in terms of resource utilization, to achieve better throughput and hence increase the performance.

Khan and Sharma (2014) modified the concept of Ant colony optimization in term of movement of the ant that is in both forward direction and backward direction. Propose an efficient technique in finding the overloaded node in minimum time. Forward Ant find overloaded node and backward ant replaced the overload virtual machine with other node. Ant finds overloaded node by moving to the forward direction. Initially ant assign the threshold value to all the node and then ant calculate the CPU utilization of each node, forward ant also maintain the pheromone table. Pheromone table contains the threshold of each node and CPU utilization of each node. If the CPU utilization is greater than the threshold then node is overloaded and ant move in backward direction to select a particular VM to place from the overloaded host. The host is checked again for being overloaded. The process is repeated until the host is considered as not overloaded by selecting host with minimum utilization and ant update the pheromone table and entered the entry of current utilization of CPU. It improves the performance in terms of throughout, response time, less energy consumption.

Xue *et al.,* (2014) presented a ACO-LB,it uses Directed Acyclic Graph and considers the average load for each virtual machine based on the ant colony algorithm to increase the

utilization rate of resources. Unlike the classical ACO where the pheromone and desired degree of transfer between two nodes are usually associated with distance etc. here the pheromone $\tau_{i,j}$ and desired degree $\eta_{i,j}$ proportional to the computing capability of nodes.

The roulette algorithm was used for solving the transfer probability problem of ants. For each task of workflow, when the probability for each virtual machine being selected is determined, a random number will be emerged in the range of 0 and 1.Then subtracting the probability of first virtual machine being selected from the number, if their difference is less than zero, the virtual machine will be selected, otherwise subtracting the probability of the next virtual machine being selected from the difference again until the subtracted result is less than or equal to 0. Therefore, the virtual machine whose probability is subtracted at last will be selected by the task. Uses Ant-Cycle model where pheromone is updated by global information. If task in the same layer are assigned to the same virtual machine, other tasks in the same same layer will be into long waiting period while one task is executing .In that case, it will lead other virtual machines which have accomplished all task on them to be in the idle state and cause the waste of resources. In order to solve this problem, once ACO-LB algorithm detects that the tasks in the same layer have selected the same virtual machineand been into a waiting period, it will select other idle virtual machines again according to probability equation.

Zhao (2014) proposed a modified algorithm based on PSO to solve the task scheduling problem in cloud computing environments by adding a cost-aware fitness function to quantify the cost of resource usage, along with the fitness function for time cost. Particle i keep track of two values: local best position $p_{LBi}$and global best position p_{GB} , both of

which are calculated. During iteration particles update themselves according to both previous and global best positions, to minimize processing time and optimized resource usage.

Nee and Gulia(2014) proposed a new load balancing algorithm using Artificial Bee Colony (ABC) for obtaining minimum makespan and less number of failure. ABC Algorithm is developed based on fixed threshold, minimum Makespan, minimum number of failure. The threshold variable determines how much load should be exchange between nodes. Then it is calculated by subtracting LightestLoad from second LightestLoad among neighbors and the load exchange happens as long as the velocity is greater than the THRESHOLD value.

According to task scheduling fault, five modifications of basic ant algorithm must be applied (Laizhi *et al* (2012) :

1) Replace the path pheromone into node pheromone to describe the handling capacity of current resource.
2) Generate ants, each ant randomly select a resource node to implement the first task
3) The meaning of pheromone is resource processing capacity.
4) According to the probability formula, ants choose the nodes to execute the next task, repeat this step until all tasks are executed.
5) Modify the relevant pheromone of each node
6) Compare with the load of the system and each ants' tasks execution time, select the tasks execution.
7) Time shortest or system efficient highest as the best option.

Moallem (2009), whose work is extended in this thesis, proposed two distributedswarm intelligence inspired load balancing algorithms;ACO and PSO scheduling algorithm. In the ACO techniques is depicted in Figure 2.5 and 2.6, each job submitted to the Grid invokes an ant and the ant searches through the network to find the best node (lightest loaded node) to deliver the job to. Ants leave information related to the nodes they have seen as a pheromone in each node which helps other ants to find lighter resources more easily and also carry the load information of the visited nodes along with themselves. Decision makingas to which node to take next, is either by looking at the load table information of nodes or they choose a node randomly by the probability of a mutation factor. At the end, the ant delivers the job to a resource and dies. In the PSO approach, As Jobs are submitted to the node in the network they go in a local Queue list of jobs in each node waiting for their turn to be executed. Each resource in the network is considered to be a particle and Nodes communicate with each other about their load information to find a better candidate to execute their workload. The amount of workload being submitted is being controlled by a threshold defined by the load difference.

```
A user submits a Job to its local resource node
An Ant is created and invoked in response to the user's requests, the job is delivered to the ant.
For each Job {
    (A)Ant starts to move from node to node for a number of steps searching for the best suitable node
    (lightest loaded node).
For each step taken by the ant {
            (i) Pheromone Laying
            Ant collects load information of the node it is visiting and adds to its history
    Updates the load information table in the visiting node
    (ii) Decision Making
    Look up Load table information of nodes
    Random selection with a probability of mutation factor.
}//end for
    (B) Ant deliver job to a particular resource and dies
}//End for
```

Figure 2. 5 : Moallem's Proposed Scheduling Pseudocode

```
Algorithm Ant (Gridlet_ID) {
  Initialize // initialization of variables
   while step < MaxSteps {
currentNode = LocalResourceID
     currentLoad ←getNodeLoadInformation
     AntHistory.add(currentLoad) // database storage
     localLoadTable.update  // database storage
     if random() < MutRate
       then nextNode = RandomlyChosenStep
//randomly generate number
       else nextNode = getLighterNodeInHistory()
     MutRate =  MutRate - DecayRate
     step = step + 1
currentNode =nextNode   // moveToNextNode
}//end while
deliverJobToNode
```

```
Algorithm getLighterNodeInHistory(){
bestNode ← currentNode
bestLoad ← currentLoad
for entry ← 1 to n {
if entry.load <bestLoad {
then bestNode ← entry.node
add(bestNode)}
else if entry.load == bestLoad
add(entry.load)
}// end for
if add() > 0
 randomlypickFrom Add()
else bestNode
}// end algorithm
```

Figure 2. 6 : Moallem's Scheduling Algorithm

Also the difference between the lightest neighbor and the second lightest neighbor so as not to burden too much load on the lightest neighbor, that it exceeds the second lightest neighbor load. Node will submit some jobs to one of its neighbors, which has the minimum

load among all. If all its neighbors are busier than the node itself, no job is submitted by the current node.

2.10 Current Fault Tolerant Scheduling Algorithms

Nandagopal *et al.*,(n.d.), proposing the checkpoint based fault tolerance mechanism with the existing Multi Criteria Resource Scheduling (MCRS) algorithm. The MCRS algorithm minimizes the response time by selecting a resource based on job requirements, job characteristics and hardware features of the resource. The fault tolerance mechanism used here sets the job checkpoints based on the resource failure rate. To compute the checkpoint interval, percentage difference between the maximum fault index and current fault index is use. Also uses failure index to evaluate the value of the resource failure rate.

Nazir et al., (2009), presented an adaptive fault tolerant job scheduling strategy for grids called CFTGS. Their strategy is checkpointing-based. It maintains the fault index of grid resources. The scheduler makes scheduling decisions according to the value of the resources fault index and response time of resources. The proposed algorithm executes more jobs successfully within the specified deadline and allotted budget, improves the overall execution time and minimizes the execution cost of grid jobs.

Jayabharathy and Parveen (2009) proposed a fault tolerant model that checkpoint tasks in a consistent state and migrate those tasks immediately to other lightly balanced nodes in an existing Intra-cluster and Intra-grid load balancing model and Also, designed a fault detector that detects the occurrence of resource failures and a fault manager that guarantees that the tasks submitted are completely executed using available resources. The performance of task execution was improved due to task migration using fault manager.

Jafar *et al*., (2009) presented two fault-tolerance mechanisms; Theft-Induced Checkpointing (TIC) relies on periodically saving the state to a stable storage and Systematic Event Logging (SEL) which log an event. It approach are suitable for applications with a need for adaptive or reactionary configuration control. The state of the execution is on a dataflow graph for efficient recovery in multithreaded application and dynamic heterogeneous system. Recovery is also done in different numbers of processors. SEL allowed for rollback at the lowest level of granularity, with a maximal computational loss of one task. The abstraction of the execution state is a dataflow graph. It shown overhead of the protocol is very small, and the maximum work lost by a crashed process is small and bounded.

Mohammad *et al.,* (2010), propose a new Genetic Algorithm that used RFOH for having reliable job scheduling in computational Grid. GIS maintains the history of the fault occurred in resources in a table called Fault Occurrence History Table (FOHT). FOHT has two columns. First column presents the history of fault occurred in the resources and second column keeps the number of job execution by resources. So the number of rows and the number of resources are equal. The fault index is incremented or decremented depending on the job status. The results demonstrate that proposed strategy reduces total execution time of jobs and decreases the probability of failure, therefore increases reliability.

Nandagopal (2010) focused on job scheduling with checkpointing based fault tolerant strategy along with the checkpoint replication service for the computational grid environment. The fault tolerant (FT) mechanism sets job checkpoint based on the failure

rate of the grid resource. When making scheduling decisions, their scheduler depends on using the fault index and the response time of resources. This fault index is maintained by taking into consideration the fault occurrence history information of the grid resource. The RRSA algorithm selects the resource for checkpoint replication. The author was able to shown that the algorithm avoids unnecessary overheads and improves the grid system performance.

Fault tolerant employed by Rebbah *et al.,* (2011) are based on error recovery by distribution and swapping techniques. The authors proposed a dynamic hierarchical model for the grid, which models the resource of the grid in a dynamic n-ary tree composed of n levels. It organizes the system in hierarchical levels. Consists of a root, set of intermediate levels according to the number of available resources, level leafs containing the resources loaded to execute jobs.

Keerthika and Kasthuri (2011) proposed algorithm that considers both system performance and user satisfaction; a new proactive fault tolerant approach for scheduling statically available meta-tasks where failure rate and the fitness value are calculated. User deadline is taken into account and the job is made to be executed within the expected deadline by assigning it to the most suitable resource. Time to Release (TTR) value is calculated for all the jobs with every available resource. The author shows that the proposed policy performs better with less TTR in the presence of failures.

Syed and Shanmugavel (2013) in their proposed scheduling algorithms resources are selected based on reliability, application execution time and cost in order to achieve optimal performance using an immune genetic algorithm. The authors presented Immune

GA based reliability optimized scheduling algorithm (IGA-RORS) aims at solving the problem of optimally allocating services on the grid to optimize the grid service reliability, deadline and cost. Check pointing is the technique employed in this algorithm, when failure occurred it migrate the task to another faster and highly reliable available resource set pool and starts the execution from the last saved state.

Arora *et al*., (2013) proposed a mechanism in which resource failure can be avoided, by proposing Dynamic Fault Tolerance on the basis of delay time of resource, by exploiting the fault handling mechanisms, and uses the knowledge of the effective speed capacity of resources and the size of task to improve fault tolerance. Information like size of task, resource speed and resource reliability are use to maintain resource history to improve application performance. Resources are selected based on the average of ratio of delay time of resource with respect to the task and size of task. Claim an improvement in application performance, resource utilization and reduction in waiting time and failure count.

Keerthika *et al*., (2013), focus was on proactive fault tolerant mechanism, where he proposed a new scheduling algorithm called Bicriteria Scheduling algorithm (BSA) algorithm, considering factors like user deadline, communication time, and failure handling mechanisms to reduced makespan. It is static heuristic, where the size of the meta-tasks, number of resources, and expected execution time of each task in each machine are known priori to execution. For scheduling job to a resource, different fitness value and a failure rate fitness value are evaluated to submit the job to a resource for execution. The authors

claim to achieves decrease in makespan and better hit rate with maximize user satisfaction and fault tolerant.

Gokuldev and Valarmathi (2013), the proposed Fault Index Based Rescheduling (FIBR) algorithm is used to reschedules the job to some other available resource when resource failure occurred. Considering, average failure time and mean failure time of the resource, when making scheduling decisions. Based on the fault index value the job is rescheduled to some other available resources with least fault index value and executed from the last saved checkpoint. The proposed kernel-level checkpoint is used to minimize execution time. Thus, claims to increases the percentage of job execution.

Elakkiyaa *et al*., (2014) proposed a genetic algorithm and a new chromosome encoding approach, to integrate the four kinds of mechanisms in a chromosome. He considered the different kinds of fault tolerant mechanisms which have different requirement for gene encoding. To balance the load each resources maintain a p number of neighboring resources which the grid is use for offloading jobs.Neighbors for each resource are formed in terms of transfer delay, as long as the neighbor is within α time of the transfer. Two possibility are assumes, sending a notification to its backup and other neighbor before shutting down or withdrawal of resources and also employed job replication strategy when failure of resource without notice, each replica is independently scheduled in a different resource to achieve fault tolerance.

In a situation where resource fault index is used during scheduling process, Amoon (2012) proposed considering resource failure rate rather than using fault index in order to select

resources that have lower probability to fail because resource can have a small fault index but it can have a high fault rate and when failure occur, the resource with high FR can lead to increase in response time and decrease in throughput. He introduces a fault-tolerant checkpointing-based system (FTCS) with a scheduling strategy, which depends on response time combined with both the failure rate and average failure time of resources when taking decision. FTCS uses the failure rate of resources to calculate the checkpoint interval for each job. He showed that it is effective when considering resource failure rate sand resource failure time over considering the resource fault index.

Surveys were carried out in fault tolerance in grid infrastructure,so as to provide guide to researchers in grid computing. Khan *et al.*, (2010) presented a performance evaluation of most commonly used fault-tolerant techniques (FTTs) in grid computing. These FTTs include retrying, checkpointing, alternate resource and alternate task. The metrics used in evaluation are throughput, turnaround time, waiting time and network delay. Garg and Kumar (2011) review the various approaches on grid fault management, fault resilience and task dependency of an application. Elaborate on various approaches in checkpoint and replication and also open issues in grid system. Fault tolerance technique in grid systems were reviewed by Altameem (2013), Space redundancy or job replication where the same job is replicated to be executed on multiple undependable resources to guard the job against a single point of failure. Time redundancy or checkpointing and rollback, the state of a running job is saved to a stable storage. This state can be used later in case of any fault to resume execution of the job instead of restating it. An adaptive technique uses both job replication and checkpointing to achieve the fault-tolerant. In addition, Altameem (2013)

review some existing scheduling algorithms and considers parameters mostly used for evaluating the performance of grid computing systems and, Supriya and Dinesh (2014) reviews various researches work that have been carried out in checkpointing algorithms on computational grid and survey on the various fault tolerance techniques and mechanism in grid computing, where the authors stated that the issues of fault detection and fault recovery is very critical and need to be address in computational grid. SunilGavaskar and Subbarao (2013) gave a survey on various fault tolerance techniques and fault management in different situations with related issues.

Kala*et al*., (2014), states that the design of fault tolerance scheduler needs to meet applications reliability requirements if performance is to be optimum. Propose a hierarchical scheduling architecture that can effectively measure both the global application and local task reliability. Hierarchical fault scheduling in computational grid is to deduct the resource failure in hierarchical manner. This hierarchical scheduling mechanism only occurs at the level of Global server and Local server; whereas fault tolerance take place at the global scheduler and fault prediction occur at the local scheduler. Showed an improvement in system performance, system reliability and reduce the speedup.

Naik*et al.,* (2012) proposed a new mechanism to improve the performance and throughput of the system by assigning tasks to a suitable and lower fault rate processor and balance the workload in each resource node. Group the resources into L discrete levels in a non-monotonic increasing order and define a closeness factor C where $0<C<1$. The algorithms tend to work well in dynamic grid environment.

Khan *et al.*,(2012) proposed median resource failure checkpointing (MRFCP) algorithms for the modification of the initially specified specific checkpoint interval. The algorithm is based on the comparison of the remaining time (RT) of the job on a resource and its median resource failure (MRF) interval whereupon it decides on the running time. MRF captures the whole failure event of the resource and whenever the system request for checkpoint, it takes all the failure events time of the resource and take its median value. This median value is then compared with the RT of the job to decide the increase/ decrease of the checkpoint intervals. When RT^{j}_{r} , is less than $MRF_{r,}$ it increases the checkpoint interval while decreasing the frequency of the checkpoint.

Prashar *et al.*,(2014) proposed an algorithm for fault tolerant distributed computation in a grid by the means of meta-heuristic Ant Colony Optimization (ACO) technique and Check-Pointing. In this algorithm, it simulates forward and backward movement of biological ants' hence two types of pheromones is use, the foraging and trailing pheromones. ACO Algorithm will initially declare the threshold levels of each node in the grid and will check the timer of ants and the load on the node, if the load on the node is below the threshold value the control will flow to the Checkpoint algorithm which receives the scheduled job from the scheduler implemented by the ACO, sets the checkpoint based on the failure rate of the resource on which it is scheduled and submits the job with checkpoints to the resource.However,with respect to the dynamism of the grid, the overloaded and underloaded informationcannot be reliable for a long time and may even causeerroneous decision-making by other ants.

2.11 Implementation Tools

2.11.1 Java programming language

The framework was built using the NetBeans IDE 8.0.1 Platform Java Programming language. Java is now one of the most widely used programming languages, mainly on networks and for web service. The major benefits of using Java as the programming language for our framework:

1. Java is object oriented, which increases the flexibility and maintainability of the programming process.
2. Multi-threading is built-in in java,which facilitates multi-threading programming.
3. Java is a language that enables the development of portable code which is often termed "platform independent" and hence makes it possible for exchange between platforms both during the development process and distribution.

2.11.2 Gridsim toolkit

The GridSim toolkit is one of the most widely used grid simulation tools. The Gridsim adopts multi-layered design architecture. In GridSim jobs can be executed at the resources in a space-shared policy or a time-shared policy. The time-shared policy (Round-robin) enables all arriving tasks to start immediately as they arrive at a resource. In the space-shared policy jobs are executed in First-come, first-serve (FCFS) order, so at each time only one application can run on one processing element. Inthis research work, Timeshared policy is used for the Algorithm scheduling to enables all arriving tasks to start executing as they arrive at the resource.

2.11.3 Reason for gridsim toolkit

The following are reasons why GridSim toolkit simulator is used to simulate and evaluate the scheduling algorithm (Moallem, 2009):

a. It allows modeling of heterogeneous types of resources.

b. Resources can be modeled operating under space-or time-shared mode.

c. Resource capability can be defined in the form of Million Instruction per second (MIPS)

d. Resources can be located in anytime zone.

e. Resources can be booked for advance reservation.

f. Applications with different parallel application models can be simulated.

g. Application tasks can be heterogeneous and they can be CPU or I/O intensive.

h. No limit on the number of application jobs that can be submitted to a resource.

i. Multiple user entities can submit tasks for execution simultaneously in the same resource.

j. Network speed between resources can be specified.

k. It supports simulation of both static and dynamic schedulers.

l. Statistics of all or selected operations can be recorded and they can be analyzed using Gridsim statistics analysis methods.

CHAPTER THREE: MATERIALS AND METHODS

3.1 Introduction

This chapter elaborates on the method applied to achieve the proposed scheduling algorithms. It shows the enhancement carried out.

3.2 The Proposed Scheduling ACO Algorithm

The ACO Algorithm steps are as follow:

1. **Job submission**: Job is submitted to the Grid.
2. **Ant Creation**: Job is delivered to an initialized ant. Ant is initialized by the Job ID.
3. **Ant Search**: Ant starts to move from resource to resource for a number of steps, to get best suitable resource (in terms of lightest loaded node and low fault rate).
4. **Pheromone Laying**: are two steps:
    - Ant collects and carries resource status information (resource load and fault rate), store it in its memory.
    - Ant updates the resource index table.
5. **Decision Making:**Ant decides which way to go according to one of the options below:
    - Resource index table
    - Choose randomly.

    This decision making is decided by the factor of mutation rate.

6. **Job Execution:** Delivers the job to the resource and dies.

Algorithm Parameter:

- An Ant ID: EachAnt as an ID for identification.
- Resource ID: ID of the initial Resource.

- Initial Resource Load Information.
- Initial Resource Fault rate
- Wander Number: The wander Number is the maximum number of steps each ant required to iterate before choosing an optimum resource. The wander number used in the simulation is five (5), i.e. each ant tour five (5) times. At the end of the iteration a resource is selected.
- Mutation Rate: The probability of ants choosing their way randomly is controlled by mutation rate, it is a constant value. As the ant is moving from one node to another, the mutation rate is decrease by a factor called decay rate as depicted in Figure 3.4.
- Decay Rate: Mutation rate decrease with a decay rate. The Decay rate is a constant value.
- And a Boolean parameter to determine if Ant has finish touring; Ant_Finish.

3.3 The Proposed SchedulingStrategyin Grid Environment

In the proposed scheduling strategy, the client through a user interface submits jobs to the grid local resource. Each resource in the grid has a Grid Resources Broker (GRB). When job is submitted to the GRB, an Ant is being created to search for the optimum resource to submit the job to. As long as GRB received a job, an ant is created. Therefore, the number of jobs submitted will be equivalent to the number of ant created because for each job submitted, it triggers an ant. For each job submitted, the ant starts searching the network to find the optimum resource to deliver the job to. As depicted in Figure 3.1when searching the network, different ant can decide to take different path. To simulate the aspect of the pheromone deposited by natural ant to find the shortest route to food source, the path pheromone

wasreplaced with resource pheromone to describe the handling capacity of current resource. Each resource in the grid as index table, which contain information about

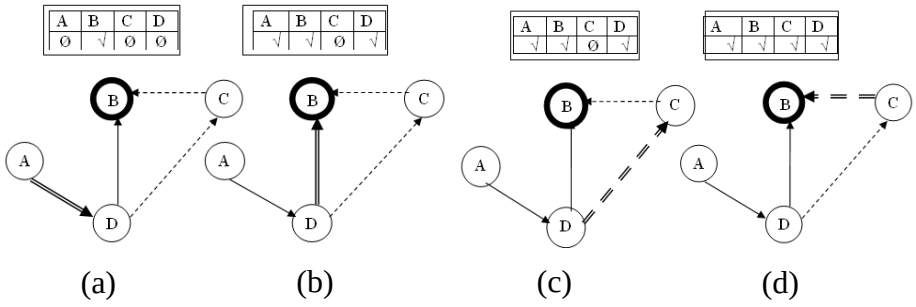


Figure 3. 1: An illustration of the ACO Model

. (a) An ant moves from A to D; (b) The ant moves from D to B; (c) Another ant moves from D to C; (d) The ant moves from C to B

other available resources in the system as depicted in Table 3.1.The index table contains two parameters:

a. Load information of the resource
b. Fault rate of the resource.

Load information of the resource is the work load on the resource (system) and fault rate is the failure rate of the resource. At the initial state,the load information of the resource is null and the number of jobsfailed are randomly generated within the range of 1 to 10 andthe number of successful jobs completed by each resource are fixed for all resource. The number of jobs failed and the number of successful job completed are used to compute the fault rate, as the algorithm start running the load and fault rate increases. In each ant tour, the ant will carry the updated information of the resource that the antvisited. Upon arrival of the ant at each resource, the following actions will be done:

1. If the resource index table does not have resource status information of the resource visited by the ant, these information will be passed to the index table.
2. If the resource index table contains resource information that is different from the information the ant has, the ant will update the information.
3. The ant will add the resource informationto its history table.

The entries of each index table is being updated by an ant who visit the resource at that time,the ant collects the resource status (the load information as well as the fault rate of the resource of the node) it is visiting and adds it in its history (memory).The ant alsoupdates the load information and fault index table in the visiting nodes. As the ant is moving from one node in the network to another it builds up statistical information about the load and fault rate of the nodes it has visited in the resources**.** The proposed algorithm is depicted in Figure 3.4.

Table 3. 1: Index Table

Resource ID	LoadInfo	FaultRate
Res 1	0.5	0.4
Res 2	0.8	0.2
Res 3	0.5	0.8
Res 4	0	0.7
.	.	.
Res n	0.5	0.5

3.4 Decision on Next Resource to Visit

Ants decide which resource to visit for their next step when searching for optimum resource that will executes the job. The ant has two choices but before an ant decide which option to pick from, a random number is generated and compared with mutation rate as

depicted in Figure 3.2, which guide the option to pickin terms of the resource current status (resource load information and fault rate). One choice is to move to arandom node with a probability of mutation rate. The other choice is to look up theresource table information in the node to choose where to go. This resource table information is the information store in the index table. When an ant decided to look up the index table information,the algorithm (Look up table) is depicted in Figure 3.3, where it compared the load and fault rate of resources to decide the nextresource to visit.

```
Randomlygenneratenum = random()
if (randomlygenneratenum< Mutation_Rate)
then
      nextNode = RandomlyChosenStep //randomly generate number
else
nextNode = getLighterNodeInHistory()
```

Figure 3.2 : Next NodeDecision Strategy

```
Algorithm getLighterNodeInHistory(){
bestNode ← currentNode
bestLoad ← currentLoad
 bestFault ← currentFault
for entry ← 1 to n{
if entry.load <bestLoad and entry.fault <bestFault{
then bestNode ← entry.node
       bestLoad ←entry.load
       bestFault ← entry.fault
     add(bestNode)}
else if entry.load < bestLoad and entry.fault > bestfault{
     then bestNode ← entry.node
       bestLoad ←entry.load
       bestFault ← entry.fault
      add(bestNode)}
else if entry.load ==bestLoad and entryfault ==bestfault{
 then
       add(entry.node)}}
if add() > 0
 randomlypickFrom Add()
else bestNode
}// end for
```

Figure 3. 3: LookUp Table Algorithm

Since natural pheromones evaporate and the strength of the trails depend on the number of ants that take that path and also the time the trail was laid, in the proposed scheduling strategy mutation rate decreases with a DecayRate factor as time passes so that the artificial ant will be

```
Algorithm Ant (Gridlet_ID){
  Initialize()  // initialization of variables
   while step < MaxSteps { // while stopping criteria are not satisfied
     currentNode = LocalResourceID
     currentLoad←getNodeLoadInformation(currentNode)
      currentFault  ← getFaultRate
     AntHistory.add(currentLoad, currentFault) //database storage
    localLoadTable.update // database storage
if random() < MutationRate
then nextNode = RandomlyChosenStep //randomly generate number
          else nextNode = getLighterNodeInHistory()
      MutationRate =  MutationRate - DecayRate
      step =  step + 1
currentNode =nextNode
}//end while
AssignCheckPoint() // depicted in Figure 3.8 pseudo code
deliverJobWithCPToNode
}//end algorithm
```

Figure 3. 4: Proposed Scheduling Algorithm

more dependent on the resource status(loadinformation and fault rate) rather than random choice. Mutation rate is a constant value of 0.5 while DecayRate is a constant of 0.2.

As depicted in Figure 3.4, after the ant as decided the next resource to visit; which is controlled by Mutation Rate. Mutation Rate decreases with a DecayRate, while the ant is still searching.

3.5 Decision Making

At the end of the iteration, that is after it has iterate for a predefined number of step (time), the ant already had information along with it in its history table, it visit this information to decide the resources to select.It chooses the lightest loaded and less fault rate node in the

table and in case of a tie the ant chooses one with an equal probability. It then submits the job to the optimum resource, after successfully submission the ant dies.

3.6 An Illustration of the Model

First, a job with ID 5 is submitted to one of the local grid resource; Res id 10, an ant is invoked with an AntID 5. The Ant in search for optimum resource will visit other nodes and try to find the best node to execute the task. It will visit other resources for a predefined number of steps. Already the ant will carry along with it the information of the Res id 10,store it in his history table. It decide to move to another resource; Res id 8, on reaching their it update the information of Res id 10 in the index table of Res id 8 (The table acts like a pheromonean ant leaves while it is moving and guides other ants to choose better paths rather thanwandering randomly in the network.)and add the resource information status of Res id 8 in its history table. It look up the information on the index table of res id 8 to decide the next resource to move to;decision making has to do with two options, as shown in Figure 3.2.An ant might either decide to look up the resource index table or they choose a node randomly by the probability of mutation factor depending on the values of "Randomlygenneratenum". It carries the information of Res Id 8 along to the next node and updates it. After completing the iteration step, it selects a resource from its list of resources in its history table and then submits the job to checkpoint handler which determine when to checkpoint and the number of times to checkpoint. Checkpointing is being done when the job is executing.

3.7 Architecture of the Proposed Fault Tolerant Technique

The interaction between the different components is depicted in Figure 3.5,whenjob is submitted to the scheduler handler by a user, it connects to GIS to getinformation of available grid resources and then requests the resources to send their current work load condition and

after that it gets the resource fault rate history of each resource from fault manager.The scheduler collects the set of appropriate machines along with their load and fault rate and it invokes the ACO techniques. The ACO algorithm elaborated in previous sections is implemented to search for the resource that meet the user requirement using the fault rate of the resource and the load on the resource, to make appropriate decisions. A Resource is selected based on the current work load and fault rate of the resources, and before forwarding the job to the selected resource, a job dispatcher dispatches the jobs to the checkpoint handler. On receiving the job from the job dispatcher, the checkpoint handler gets the number of successful job completion and number of failed job of the resource from the fault index handler on which it is scheduled and sets the number of time to checkpoint and the checkpoint interval based on the failure rate of the resource. Then the checkpoint handler submits the job along with the checkpoints to the selected resource. The checkpoint repository receives partially executed result of a task from a grid resource in the intervals specified by the checkpoint handler. It maintains grid tasks and their checkpoint table which contains information of partially executed tasks by the grid resources.For a particular job, the checkpoint repository discards the result of the previous checkpoint, when a new value of the checkpoint result is received. When a particular job is completed, its entity will be removedfrom the checkpoint result table. The Fault Handler, a class in Gridsim monitors the status of the resources available in the grid at regular intervals. If any failure occurs, it is reported to the checkpoint handler. Checkpoint handler updates the fault index handler to increment the failure index of the grid resource.

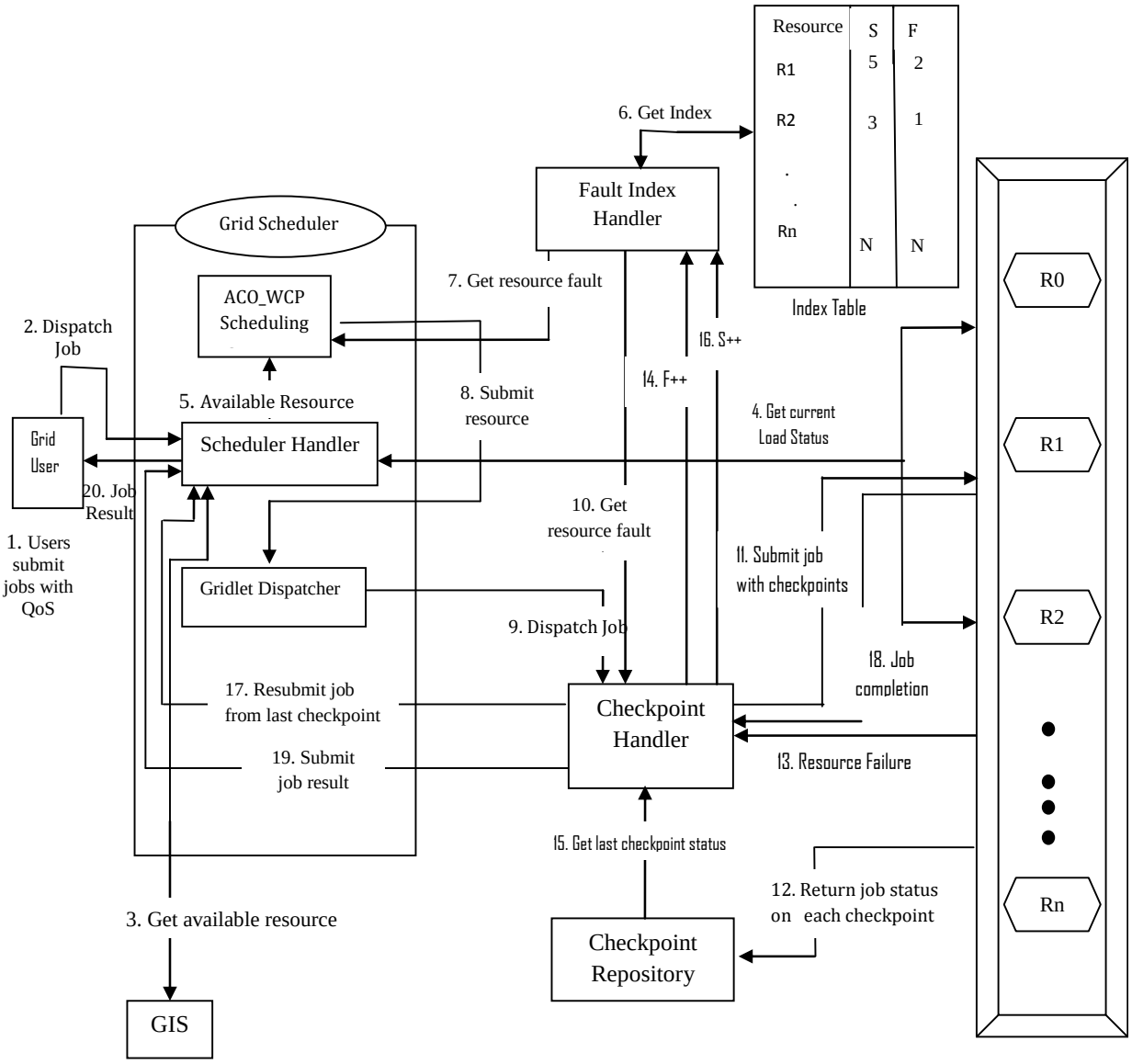


Figure 3. 5: Architecture of the proposed fault tolerance in grid computing

The fault index value suggests the rate of tendency of resource failure, the lesser the fault index value, the lesser the failure rate of the resource and the higher the fault index value, the higher the failure rate. Checkpoint handler then queries the checkpoint repository to obtain latest checkpoint files of the executed jobs on the failed resource and rescheduled the jobs along with last checkpoint status.On the successful completion of the job, the checkpoint handler receives the job completion message from the grid resource and updates the fault index handler to

increment the success rate of the resource The Fault Index Handler maintains a fault index history of the grid resources, which indicates the failure rate of the resource. To update and maintain the fault index of a grid resource, the fault index handler uses the algorithm described below to take decision:

Algorithm : Fault Index Handler

- IF the checkpoint handler receives the job completion message from resource THEN
  - Send a message to the fault index handler to increment success index of the resource.
  - Submits finished job to the scheduler.

END IF

- IF the checkpoint handler receives the failure message from the resource monitor THEN
  - Send a message to the fault index handler to increment the failure index of the resource that fails to complete the assigned job.
  - Send a message to the checkpoint repository, to check checkpoint status of this job.
- IF the checkpoint result of the job exists in the checkpoint repository THEN
  - Submit the last checkpoint data received, to the scheduler handler for rescheduling.
  - Exit

END IF

- IF the checkpoint result of the job does not exist in the checkpoint repository THEN
  - Resubmit the job to start from scratch to the scheduler handler for rescheduling.
  - Exit

END IF
END IF

Figure 3. 6 : Algorithm of the Fault Index

Checkpoint Algorithm

```
S1: set checkpoint time and interval for a particular job
S2: submit the job for execution
While job is running till the end
S3: if currentTime ≥ checkpointTime
        Then
            Update checkpoint interval time
            Increment number of time checkpoint
            Checkpoint current process state
        Else not time for checkpoint
```

Figure 3. 7: Checkpoint Algorithm

3.8 Checkpointing Tool Implementation

Checkpointing is combination of two activities; save the running data and restore it after getting suitable resource. Captured states and data of a running process include registers containing the address, variables, memory spaces keeping source codes, libraries, data structures, files containing data with a large size and other data. Anapplication-level checkpointing tool is adopted, which is directly implemented within the application source code. That is, the application contains source code that saves and restores critical program variable to and from stable storage. The checkpointing of an object-oriented program is considered, in which the state of the program can be recovered from the contents of the fields of the objects. In this context, checkpointing amounts to recursively traversing the objects and recording the local state of each one.It is essential to identify what is to be checkpoint. The implementation consists of aGridletCheckpointing which specifies the logic condition that must be provided by each object to be checkpointed. When it is time to take a checkpoint, the application writer is expected to explicitly call the function CheckpointRepository() whenever a checkpoint is required,GridletCheckpointing object, drives the checkpointing process.The checkpoint file use in this experiment is a Hash function. Hashing is a technique of storing values and searching for them in tables. Table is a set of table entries, (K; V), each containing a unique key K, and a value (information) V. Each key uniquely identifies its entry. For simplicity the assumption made is that checkpoints are written to this hash table.Each checkpoint contains only the pages that have been modified since the previous checkpoint.Here, Hash table keyed is used, to keep track of the necessary objects.It used a hash table keyed on identity hash code of the objects.

This hash table maps from the object to a pair: <object, address>. Object is a reference back to the object; address refers to the address of the object in the output image.

The resource failure rate (*FR*) is used to determine the checkpoint interval and the number of checkpoints instead of using the resource fault index. Resource failure rate is used to represent failure history of resource. The formulae used to compute the number of checkpoint and checkpoint interval is gotten from Amoon (2012). Equation 3.1calculates the fault rates of the resources and equation 3.2 calculates the number of times to checkpoint a particular job when it is running and equation 3.3 calculates the checkpoint interval time when a job should be checkpoint.

$$Failure\ Rate(FR) = \frac{Failed\ Job}{Number\ of\ Jobs\ Submitted} \quad \ldots (3.1)$$ Amoon (2012)

$$CheckpointNumber = R_t * FR \quad \ldots (3.2)$$ Amoon (2012)

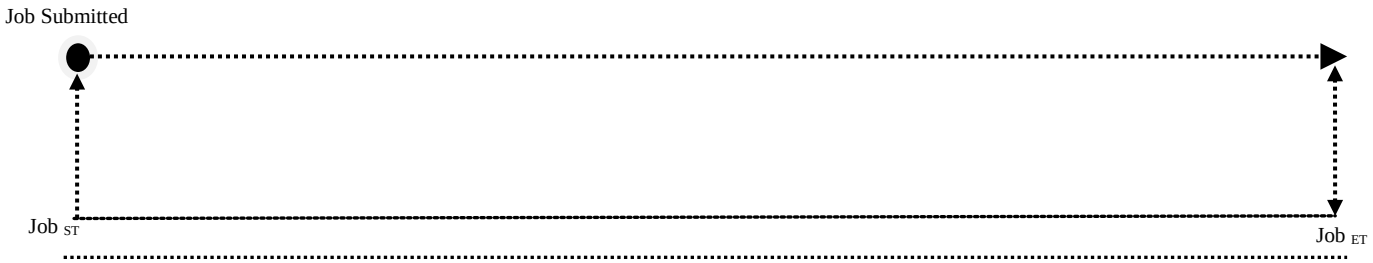
$$where\ R_t = Response\ time$$

$$checkpointInterval = \frac{R_t}{checkpointNumber} \quad \ldots (3.3)$$ Amoon(2012)

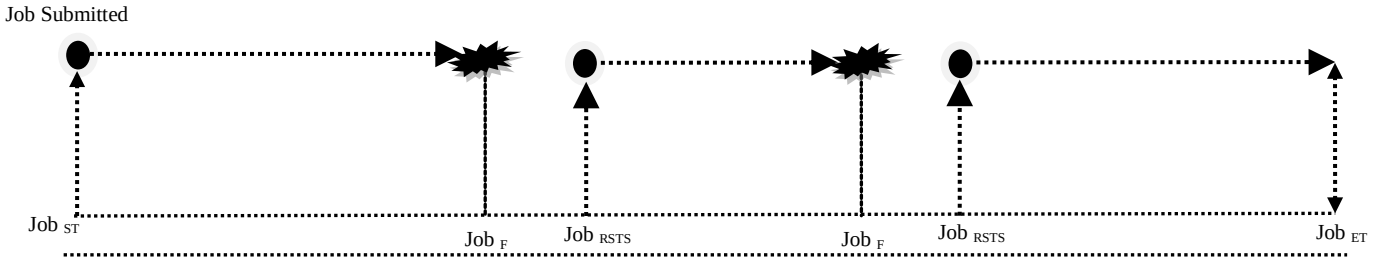
3.9 An Illustration of the Rollback Recovery Analysis

In a conventional system, when a failure occurs, an application has to be restarted on another grid resource from the beginning. A technique to avoid restarting the application from the beginning is the rollback recovery, which is based on the concept of checkpoint. The checkpoint mechanism periodically saves the application's execution state to stable storage. Hence, whenever a failure interrupts a volunteer computation, the job can be resumed from its last successful state, thereby improving the QoS requirement of the users. An Illustration of the proposed scheduling algorithms is depicted in Figure 3.8. The first diagram shows a

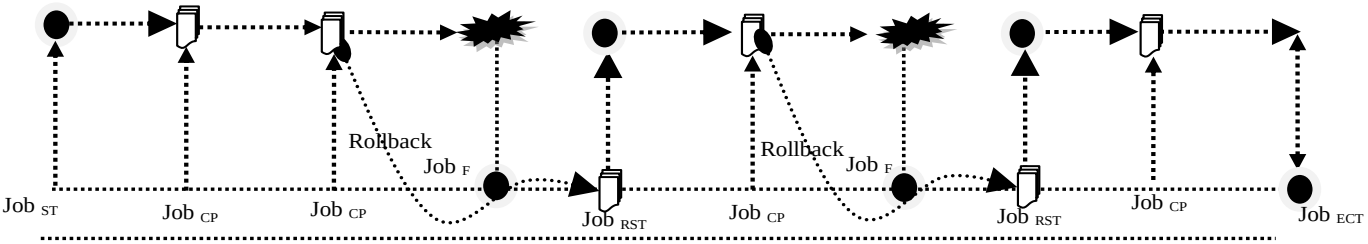
normal environment where job failure does not exist, when the job is submitted it run to completion. Although with the grid environment where resource failure is guaranteed, the dynamism is put into consideration. The scenario is as follows: In Moallem algorithm, discussed in chapter 2, when a resource failure occurs, the scheduler needs to reschedule the job which starts its execution from scratch, leading to an increase in the execution time of the job. To improve the performance of Moallem proposed algorithm (Moallem, 2009), a checkpoint based fault tolerance and recovery strategy is integrated into the existing algorithm, the return time will be improved considerably that is useful for the environments with high source fault rate. In the new proposed algorithms, given in figure 3.4, the job execution status is preserved in the form of checkpoints. When a resource failure occurs, the job is restart exactly from the last saved checkpoint, thereby eliminating the need to start the job from scratch. Hence, the execution time of the job is reduced.



Normal Scenario (No failure, No Checkpointing)



Molleam strategy: ACO (Failure, No Checkpointing)



Proposed: ACO_CP (Failure,Checkpointing)

Job_{ST} : Job execution start time
Job_{CP} : Job checkpoint file created time
Job_{RST} : Job restart time from last checkpoint
Job_{RSTS}:Job restart time from stratch
Job_{ECT} : Job execution end time when restarted using checkpoint
Job_{F} : failure notification Time
JobET:Job execution end time

Figure 3. 8: An illustration of the recovery analysis

The submitted job starts its execution at time Job_{ST}(Job start time)and finishes its execution at time Job_{ET}(Job execution end time)in a normal environment, i.e., the execution time of the job is given in equation (3.4).

$$Execution\ Time = (Job_{ET} - Job_{ST}) \quad \ldots \quad (3.4)$$

And without considering checkpoint, if failure occurs at time Job_F(Job failure notification time) then the job restarts its execution from beginning at time Job_{RSTS}(Job restart time from scratch)and finishes at time Job_{ET} (Job execution end time), i.e., the total execution time of the job as given in equation (3.5) is the summation of the following: Job_F - Job_{ST}; Job_F - Job_{RSTS}; Job_{ET} - Job_{RSTS};

$$Total\ Execution\ Time = (Job_F - Job_{ST}) + (Job_F - Job_{RSTS}) + (Job_{ET} - Job_{RSTS}) \ldots (3.5)$$

If the job execution status is maintained in the form of a checkpoint file, and the last created checkpoint is at time Job_{CP}(job checkpoint file created time) which is before Job_F, then the job can be restarted from the last checkpoint at time Job_{RST}(Job restart time from last checkpoint). Also the total completion time as given in equation (3.6) will be summation of Job_F - Job_{ST}; Job_F - Job_{RST}; Job_{ET} - Job_{RST};

$$Total\ Execution\ Time = (Job_F - Job_{ST}) + (Job_F - Job_{RST}) + (Job_{ECT} - Job_{RST}) \ldots (3.6)$$

3.10 Feature of Gridsim Resource Failure Modified

The gridsim toolkit failure classes uses RegionalGisWithFailure, it involves regional GIS.A regional GIS is a region of the network topology. In it there can be multiple nodes, then one or more routers that are connected to one or more resources.For the purpose of comparison and analysis, to fit in with Molleam proposed scheduling algorithms, GIS was extended to include features of failure that is used in RegionalGisWithFailure. Using RegionalGisWithFailure will

not fit in with Molleam scheduling algorithms sincethe failure parameter that determines how many resources will fail, also assume each resource has multiply machines, so it decides how many machines for each resource will fail, when and how long the resource will fail, are generated using HyperExponential, the number of resourcesto fail is generated randomly but for experimental purpose, the number of resources to fail is fixedand also a resource involve one machine for the purpose of accurate comparison and analysis with the two scheduling algorithms.

3.11 System Model

For experimental purpose, the resources are assumed to be homogeneous so as to allow us to make efficient comparison of the proposed ACO_WCP with ACO proposed by Moallem (2009) to show the efficiency of the algorithms despitethe existence ofhomogeneous characteristics. It is assumed that the grid consists of one computing node (machine) per resource and 2 Processing Elements (PE). And also Network speed among the computing nodes is also the same as shown in Table 3.2.

Table 3. 2: Grid Resource Characteristics

Grid Resource Characteristics	
Number of machines per resource	1
Number of PE per machine	2
PE ratings	50 MIPS
BandWidth	5000 B/S

3.12 Application Model

The application model, assume that jobs which are submitted to the grid are independent tasks with no required order of execution. The jobs are of the same computational size, same input files size requirements for experimental purpose. The computational requirement (Job length) of each job is presented in Millions Instructions (MI). In GridSim, jobs are created and their requirements are defined through Gridlet objects. A Gridlet is a package that contains all the information related to the job and its execution management details such as the job length (in Million Instructions(MI)), the size of input files, and the job originator (user information). The characteristics of the Gridlet used in the simulation sent to the grid to compare the performance of different algorithms are shown in Table 3.3.

Table 3. 3: Gridlet Characteristics

Gridlet Characteristics	
Length	0 – 50000 MI
Input file size	100 + (10% to 40%)
Output file size	250 + (10% to 50%)

3.13 Performance Evaluation Criteria

In this section, performance evaluation criteria which are used to evaluate the performance of the proposed algorithm are defined. The criteria include makespan, average turnaround time and throughput.As the gridlets and topologies are generated randomly, although every simulation yields roughly the same result, each single simulation is different from another one; thus, an average of 1000 runs were made using batch file in order to simulate realistic conditions and the average were used to plot graph.

$$Makespan = finish\ time\ of\ the\ last\ task - start\ time\ of\ the\ first\ task \quad \ldots (3.7)$$

$$Throughput = \frac{number\ of\ gridlets}{time\ to\ complete\ all\ the\ gridlets} \quad \ldots (3.8)$$

$$Turn\ Around\ Time = gridlet\ Execution\ time - gridlet\ start\ time \quad \ldots (3.9)$$

$$Average\ Turn\ Around\ Time = \frac{\sum_{i=0}^{n} Turn\ Around\ time}{n} \quad \ldots (3.10)$$

Where $n = number\ of\ gridlets\ and$

$gridlets = jobs$

But the Turn Around time in this scenario where failure is considered, equation 3.8 will not hold. Therefore equation 3.5 was used in moallem's simulation since no fault mechanisms were considered, it starts again from scratch when failure occurred and the new proposed algorithm used equation 3.6 where it will start from the last checkpoint.

How Percentage Difference were Evaluated:

Makespan:

$$\frac{MoallemMakespansimulationResult - ProposedMakespanSimulationResult}{MoallemMakespanSimulationResult} * 100$$

Summation of Moallem simulation result in table 4.1 gave 3982555.222 and summation of the proposed algorithm simulation result gave 3450830.781. Makespan is minimized by:

Therefore

$$\frac{3982555.222 - 3450830.781}{3982555.222} * 100 = 13.35134$$

$$=13\%$$

Throughput:

$$\frac{Proposed\ Throughput\ Simulation\ Result - Moallem\ throughput\ simulation\ Result}{Moallem\ throughput\ Simulation\ Result} * 100$$

Summation of Moallem simulation result in table 4.2 gave 0.043675225 and summation of the proposed algorithm simulation result gave0.048833661. Throughput is maximized by:

Therefore $\frac{0.048833661-0.043675225}{0.043675225} * 100$

$= 11.8109$

$=12\%$

Average Turn around (ATA):

$$\frac{Moallem\ ATA\ simulation\ Result - Proposed\ ATA\ Simulation\ Result}{Moallem\ ATA\ Simulation\ Result} * 100$$

Summation of Moallem ATA simulation result in table 4.3 gave 2700410.764 and summation of the proposed algorithm simulation result gave 2374125.463. Average Turnaround time is minimized by:

$$\frac{2700410.764 - 2374125.463}{2700410.764} * 100 = 12.0828026$$

$=12\%$

By varied the resources while the Gridlets are kept constants the percentage difference is shown below:

Makespan:

$$\frac{Moallem\ Makespan\ simulation\ Result - Proposed\ Makespan\ Simulation\ Result}{Moallem\ Makespan\ Simulation\ Result} * 100$$

Summation of Moallem simulation result in table 4.4 gave 12627235.85 and summation of the proposed algorithm simulation result gave 10322362.33

$$\frac{12627235.85 - 10322362.33}{12627235.85} * 100 = 18.2531913 = 18\%$$

Throughput:

$$\frac{Moallem\ Throughput\ simulation\ Result - \ Proposed\ ThroughputSimulation\ Result}{Moallem\ Throughput\ Simulation\ Result} * 100$$

Summation of Moallem's throughput simulation result in table 4.5 gave 0.106535198and summation of the proposed algorithm simulation result gave 0.125706034 $\frac{0.125706034 - 0.106535198}{0.106535198} * 100$ = 17.9948 =18%

Average Turn Around Time:

$$\frac{Moallem\ ATA\ simulation\ Result - \ Proposed\ ATA\ Simulation\ Result}{Moallem\ ATA\ Simulation\ Result} * 100$$

Summation of moallem simulation result in table 4.6 gave 6551769.266 and summation of the proposed algorithm simulation result gave 5650128.807

$$\frac{6551769.266 - 5650128.807}{6551769.266} * 100 = 13.7617859$$

$$= 14\%$$

CHAPTER FOUR: ANALYSIS OF RESULTS

4.1 Introduction

This section shows a comparison of the simulated algorithms when using ACO_WCP (with checkpoint) and ACO algorithm (without checkpoint) in distributed grid environment. In ACO algorithm, when a resource failure occurs, the scheduler needs to reschedule the job which starts its execution from scratch, leading to an increase in the execution time of the job. But in ACO_WCP, the job execution status is preserved in the form of checkpoints. When a resource failure occurs, the job is restarted exactly from the last saved checkpoint, i.e., eliminating the need to start the job from scratch. Hence, the execution time of the job is reduced. ACO_WCP algorithm is simulated using GridSim toolkit to verify the improvement of the proposed approach over ACO proposed by Moallem (2009) scheduling strategies. The experimental results are analyzed based on performance criterions used.

4.2 Results

In order to evaluate the performance of the proposed algorithm, a set of experiments were conducted to measure makespan, Average turnaround time and throughput. The result of the proposed scheduling algorithm with Checkpoint (Withfailure) is compared with Moallem's proposed algorithm (Withoutfailure). In the simulation,scheduling experiments is performed by keeping the resource constant and setting different values to the number of jobs; the number ofGridlet is varied from 100 to 3700 and the length of each gridlets is 200,000 MIs at each step and investigate. For each job submitted, an ant is created and ant is assumed to travel from one resource to another in an increment of one (1) time step, each ant is assumed

to take 5 steps and stop when this criterion is reached. It is clear that the efficiency is improved, since the tour of the ants consider both the resource load and fault rate of the resource. This increases the chance of each ant to find a resource with low fault tendency.

The tables 4.1, 4.2 and 4.3 show the simulation results of ACO_WCP (Withfailure) and ACO(Withoutfailure) algorithms. The Figures 4.1, 4.2 and 4.3depicts the makespan, average turnaround timeand throughput. The makespan is good when there is a decrease in makespan, so therefore the algorithm is improved if the makespan decreased compare to the proposed Moallem algorithm. Likewise the average turnaround time, the average turnaround time is good when there is a decrease in the proposed algorithm, when comparedwith Moallem's algorithm. And throughput is good when it is maximized. Therefore a decrease in makespan and average turnaround time and an increase in throughput is an improvement. As the size of the number of Gridletsis increased, so the numbers of jobs failure increases because a particular resource at the time of failure might have more jobsin execution when it failed.The intention here, is to find its effect on the makespan, as number of gridlet increases the proposed algorithms tend to be better, this is expected since the proposed scheduling algorithms is keeping track of the process state of the entire jobexecuting in each resource at a particular interval of time, it tends to save a lot of execution time by avoiding the situation of starting from starch because haven more than one job executing and when there is resource failure, it affect the whole jobs in that resource. In Figure 4.2, as the number of Gridlets increases, the proposed scheduling algorithms tend to have better result. At the initial state, that is when number of Gridlets is 100, Moallem as a good result, even when Gridlets number is 500, this is because few jobs and many resources, resources might be executing few jobs when failure occurs. As the number of Gridlets increases from 500 to

900, the two graph meet when the number of Gridlets is at 900 and also at 1300. As the size of Gridlets increases,the job failure increases because a particular resource at the time of failure might have more jobs in execution when it failed. The proposed scheduling algorithm tends to improved better than Moallem's algorithm, with a percentage difference of13%, which is adecrease in makespan.Figure 4.2, which depict Throughput for varied Gridlets, the goal here is maximization, as Gridlets size increases the graph produced a better result, at size 100 and 500 Moallem's result was better, this was as a result of few jobs executing at that time, at size 900, the graph meet and also when the size of Gridlets is at 1300. The graph start separating as the sizes of the Gridlets increases. From Figure 4.2, increment in the Gridlets size improved the new proposed algorithm with a percentage increase of 12% when compared with Moallem's algorithm.12% decrease in Average Turn Around time was observed when compared with Moallem simulation, as shown in Figure 4.3. The size of the Gridlets improved the average turnaround time of theproposed scheduling algorithms. This improvement was as a result of checkpoint and selection of resourceswithlow failure tendency.

Table 4. 1: Average Makespan table for varied Gridlets

NumResource	NumGridlet	WithOutFailure	WithFailure
100	100	48569.8266	82581.19148
100	500	156705.023	159281.0954
100	900	235171.492	235238.1332
100	1300	306692.039	298183.8147
100	1700	368990.225	347092.3771
100	2100	445146.758	383579.4808
100	2500	493102.208	423205.2016
100	2900	572050.794	467180.1945
100	3300	643413.86	505964.9605
100	3700	712712.996	548524.3321

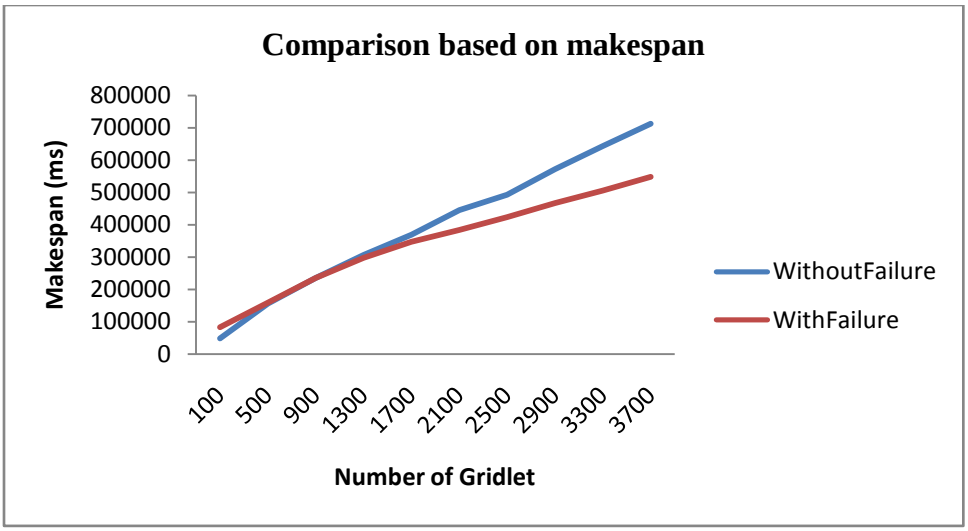


Figure4. 1: Graph of average Makespan for varied Gridlets

Table 4. 2: Average Throughput table for varied Gridlets

NumResource	NumGridlet	WithOutFailure	WithFailure
100	100	0.002137812	0.001295921
100	500	0.003261955	0.003220209
100	900	0.003888949	0.003908796
100	1300	0.004304247	0.004431326
100	1700	0.004655346	0.004955437
100	2100	0.004779502	0.005525678
100	2500	0.005112216	0.005947022
100	2900	0.005121511	0.006234242
100	3300	0.005176597	0.0065532
100	3700	0.00523709	0.00676183

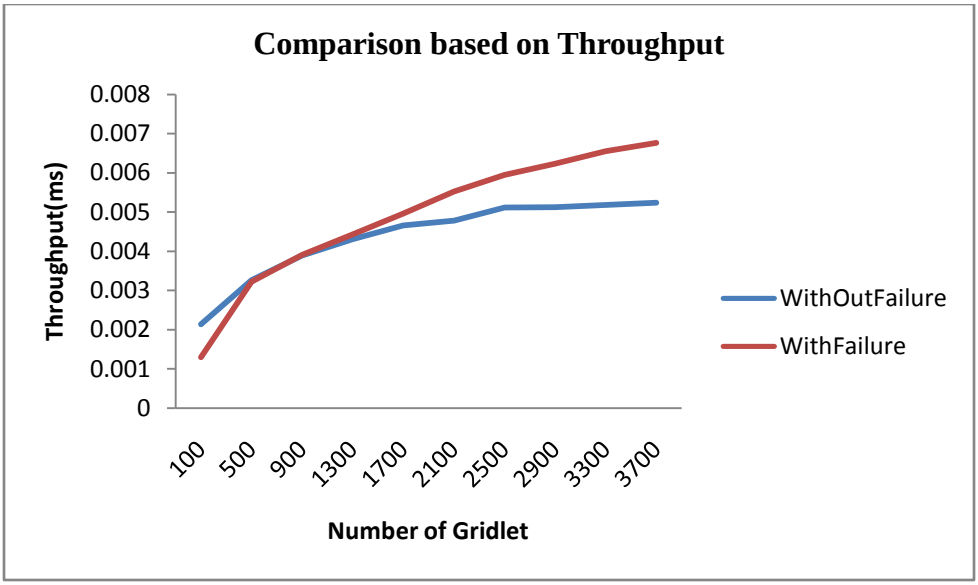


Figure 4. 2: Graph of average Throughput for varied Gridlets

Table 4. 3: Average Turnaround time for varied Gridlets

NumResource	NumGridlet	WithOutFailure	WithFailure
100	100	25191.8559	32851.01982
100	500	72079.09801	79757.59004
100	900	128938.2533	134389.9038
100	1300	185247.1481	186865.8473
100	1700	241360.751	228997.5128
100	2100	297892.2344	262358.6086
100	2500	352209.5551	298553.1052
100	2900	407783.3016	339709.521
100	3300	466519.0582	383153.9393
100	3700	523189.5081	427488.4148

Experiments werealso carried outby keeping the number of Gridlets constant and varied the number of resources. As given in the Tables 4.4 for Makespan, 4.5 for throughput and 4.6 for average turnaround time.

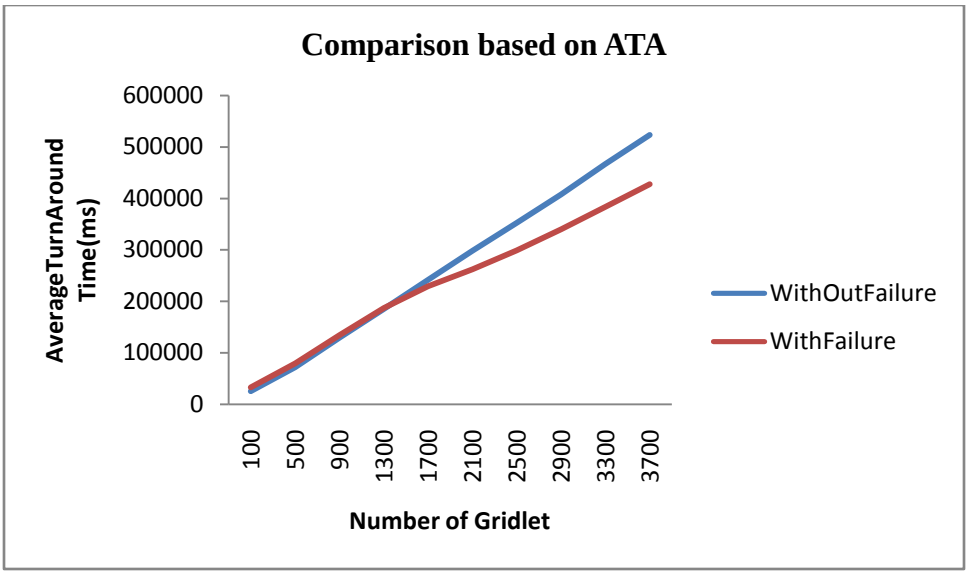


Figure 4. 3: Graph of Average Turnaround time for varied Number of Gridlets

Table 4. 4: Average Makespan time for varied Resources

NumGridlets	NumResources	Withoutfailure	Withfailure
3000	50	5333758.145	4193551.119
3000	250	1032621.551	915672.9539
3000	450	735204.4272	635026.9101
3000	650	608917.605	522718.7224
3000	850	547930.762	462654.4447
3000	1050	510076.0521	425486.6746
3000	1250	471787.1408	384175.1397
3000	1450	437496.6908	366323.4776
3000	1650	416411.9805	324779.3881
3000	1850	384378.3534	330142.321
3000	2050	379767.6597	317569.9529
3000	2250	367627.454	311514.3109
3000	2450	371217.0904	293752.8611
3000	2650	357757.9686	282355.1981
3000	2850	341346.5343	284203.798
3000	3050	330936.4383	272435.0624

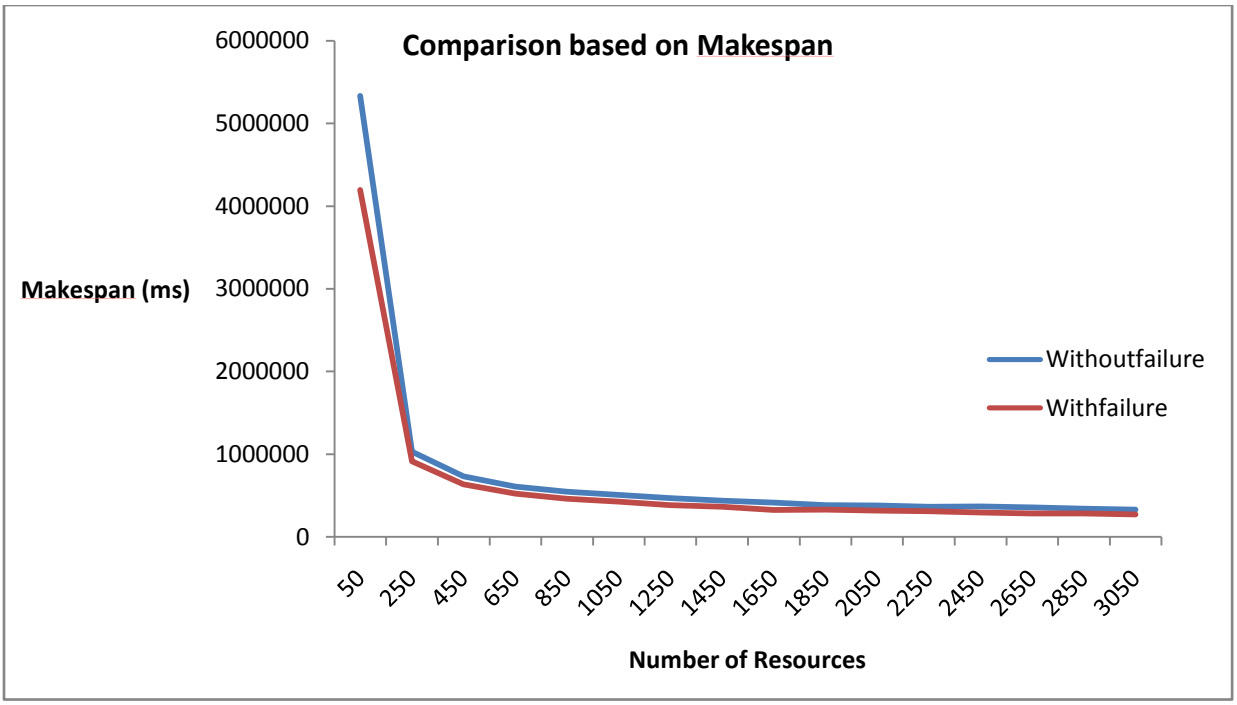


Figure 4. 4: Graph of Average Makespan time for varied Number of Resources

Table 4. 5: Average Throughput time for varied Resources

NumGridlets	NumResources	Withoutfailure	Withfailure
3000	50	0.000578341	0.000723232
3000	250	0.002929033	0.003296531
3000	450	0.004154446	0.004769876
3000	650	0.005048586	0.005824279
3000	850	0.005629315	0.006571081
3000	1050	0.006114395	0.007189014
3000	1250	0.00665545	0.007918572
3000	1450	0.007119166	0.008371468
3000	1650	0.007432594	0.008859062
3000	1850	0.008134158	0.009326724
3000	2050	0.008270621	0.009682522
3000	2250	0.008510047	0.009844879
3000	2450	0.00849208	0.010470581
3000	2650	0.00878381	0.010836466
3000	2850	0.009138704	0.010826396
3000	3050	0.009544452	0.011195351

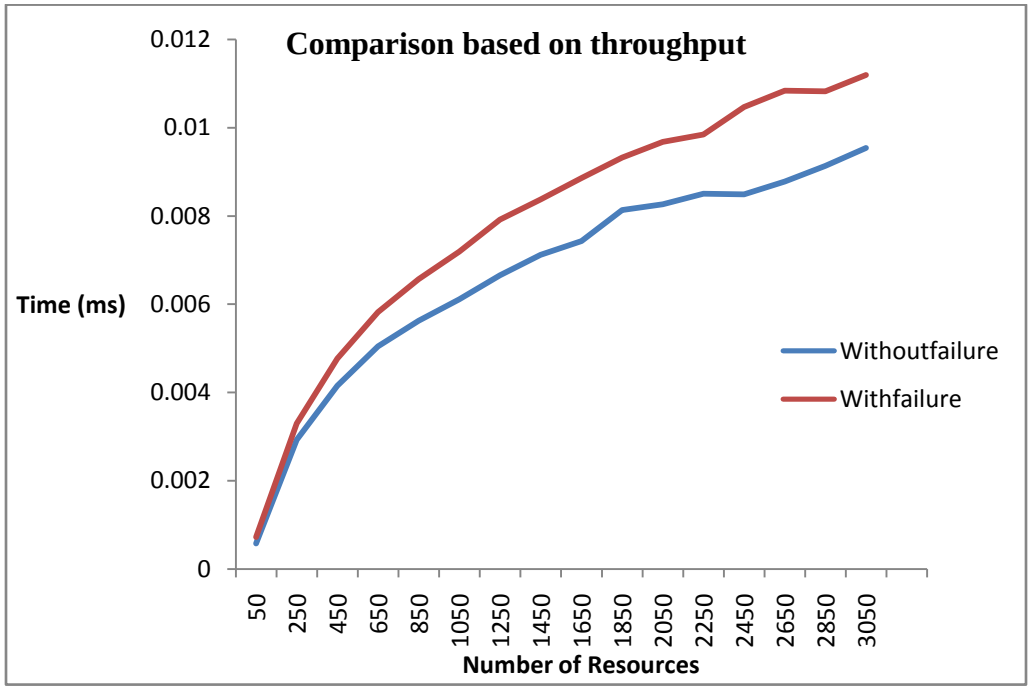


Figure 4. 5: Graph of Average Throughput time for varied Number of resources

Table 4. 6:Average Turnaround time for varied Resources

NumGridlets	NumResources	Withoutfailure	Withfailure
3000	50	3807268.06	3026939.176
3000	250	631508.8115	569764.0766
3000	450	345302.0931	324526.2434
3000	650	243015.8456	232252.3372
3000	850	193772.7683	186910.5638
3000	1050	164348.6584	159492.8966
3000	1250	145337.533	141939.9795
3000	1450	133213.1211	130883.2574
3000	1650	123799.4381	121695.203
3000	1850	118449.8406	116649.3752
3000	2050	113883.6945	112504.8401
3000	2250	110544.2414	109215.7295
3000	2450	108441.4055	107411.0636
3000	2650	106048.3362	104947.0919
3000	2850	104075.5757	103099.5684
3000	3050	102759.8428	101897.4044

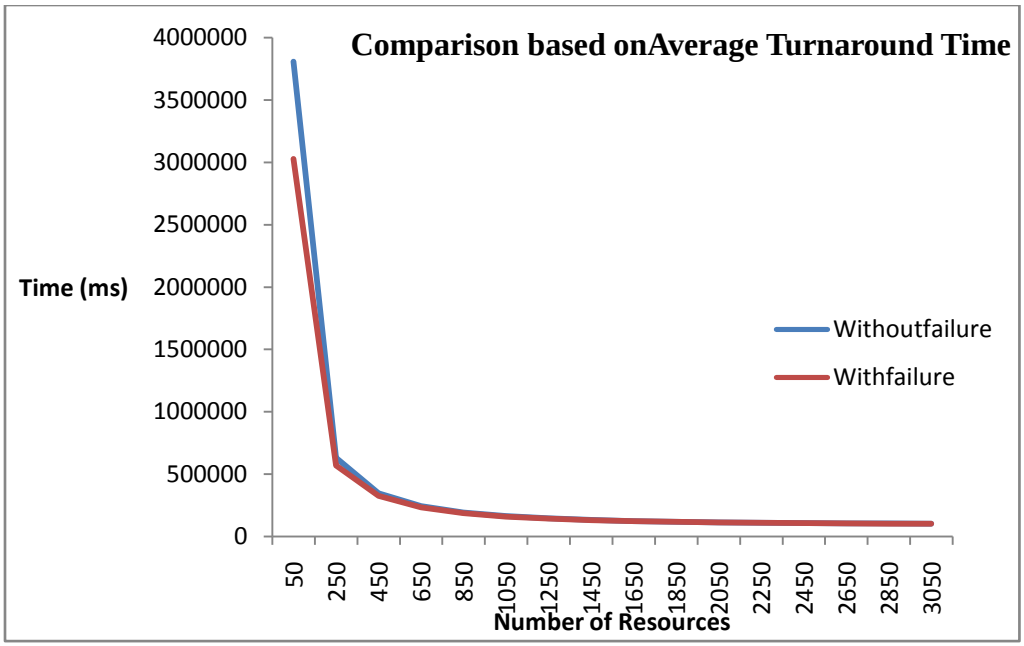


Figure 4. 6: Graph of Average Turnaround time for varied Number of resources

Table 4.4, 4.5 and 4.6 gives the results obtain as the number of Gridlets is kept constant and varied the resources.In this experiment, 3000 jobs are sent to the Grid with varying numberof resources from 50 to 3050. Figure 4.4, which depicts the makespan result simulation,since the size of Gridlets is constant for all resources.As there is an increase in resources, it can be observed that there is a decreasing exponential effect on the execution time. The algorithm achieved 18% reduction in makespan, 18% maximization in throughput and up to 14% reduction in average turnaround time.

CHAPTER FIVE: SUMMARY, CONCLUSION AND RECOMMENDATION

5.1 Summary

This thesis describes the fault tolerance used to enhance theperformance of the resource scheduling algorithm in grid computing environment, by introducing ACO with checkpointing mechanism. The ACO does not address the fault tolerance requirements explicitly. ACO_WCP uses ACO algorithm, and builds fault tolerant solutions around it, to support checkpoint based fault tolerance. By adding fault tolerant features within the scheduling approach, the overall performance of the grid system is improved.

The simulated resultsdemonstrate that the proposed strategy effectively schedules the grid jobs even in thepresence of failures. It is observed from the experiments that the fault tolerance basedresource scheduling strategy provides better results than the resource schedulingalgorithm without fault tolerance in terms of various performance metrics, such as makespan, average turnaround time and the number of jobs completed. The proposed strategy has shown tangibleimprovements in satisfying the user's QoS requirements for the submitted jobs in a faulttolerant way, in spite of the highly dynamic nature of the grid.

5.2 Conclusion

Grid computing is a high performance computing environment to solve large-scale computational demands. Checkpointing is defined as a designated place in a program at which normal processing is interrupted specifically to preserve the status information necessary to allow resumption of processing at a later time. By periodically invoking the checkpointing process, one can save the status of a program at regular intervals. If there is a failure one may

restart computation from the last checkpoint thereby avoiding repeating computation from the beginning. The process of resuming computation by rolling back to a saved state is called rollback recovery.In line with the objectives, the first objective was proposal of a fault tolerant scheduling mechanism that will be incorporated into an ACO scheduling algorithm. This was achieved by extending Moallem's ACO scheduling algorithm by incorporating fault tolerant mechanism. The fault tolerant mechanism used here is the checkpointing and rollback recovery mechanism. The second objective was to improve the user Quality of Service requirements, which was done by implementing and evaluating the proposed fault-tolerant ACOscheduling algorithm. The result of the two compared algorithms shows that the proposed algorithmachieved up to 13% reduction in makespan, 12% maximization in terms of throughput and 12%reduction in ATA when the Gridlets are varied and the resources are kept constant. Also when the Resources are varied and Gridlets are kept constant, the proposed algorithm achieved 18% reduction in makespan, 18% maximization in terms of throughput and up to 14%reduction in ATA. The last but not the least objective was to reduce the selection probability of resources with more fault occurrence history. Amoon (2012) in his research work had shown that rather than usingfault index of resources the failure rate of resources is better, itimproved performance. With the inclusion of fault history of resources into the ACO scheduling algorithm, the tendency of selecting a resource with high failure rate is minimize.

5.3 Recommendation

In future research some more factors like jobs with a deadline, network delay, QoS (Quality of Service) requirements should be specifiedand checkpointing shouldinvolve an external file. Checkpoint latency should be considered when reading and writing to a file.

REFERENCES

Abdul Azeez, I., and Haque, S. (2012). Resource Management in Grid Computing: A Review. *Greener Journal of Science Engineering and Technological Research (2276-7835), 2(1)* , 024-031.

Ali, A. D., Belal, M. A., and Al-Zoubi, M. B. (2010). Load balancing of distributed systems based on multiple ant colonies optimization. *American Journal of Applied Sciences, 7(3)* , 428-433.

Altameem, T. (2013). Fault Tolerance Techniques in Grid Computing Systems. *International Journal of Computer Science and Information Technologies, 4(6)* , 858-862.

Amoon, M. (2011). A Development of Fault-Tolerant and Scheduling System for Grid Computing. *GESJ: Computer Science and Telecommunications, (1512-1232), 3(32)* .

Amoon, M. (2012). A Fault Tolerant Scheduling System Based on Checkpointing for Computational Grids. International Journal of Advanced Science and Technology, 48.

Arora, G., Rani, S., and Singh, P. (2013). Dynamic Fault Tolerance in Desktop Grids Based On Reliability. *International Journal of Computers and Technology, (2277-3061), 11 (4)* .

Barboni, S. (2011). Assessment of dependability scenarios in large-scale grids, using GridSim toolkit. *Master's Thesis* . Institute of Computer Science of the Ludwig Maximilians Universit.www.nm.ifi.lmu.de/pub/Fopras/barb11/PDF-version/barb11.pdf

Barde, C. R., Kasar, S., Nikam, S., Shelar, S., and Wagh, N. (2014). ANT Algorithm for Load Balancing Problem in FTP Server. *International Journal of Emerging Technology and Advanced Engineering, 4(2)* .

Batrashev, O. (2007). Fault tolerance method for MPI Fortran Programs. *Tartu University Faculty of Mathematics and Computer Science Institute of Computer science* .

Bronevetsky, G., Fernandes, R., Marques, D., Pinga, K., and Stodghill, P. (2003). Automated application-level checkpointing of mpi programs.

Bronevetsky, G., Fernandes, R., Marques, D., Pingali, K., and Stodghill, P. (n.d.). Recent Advances in Checkpoint/Recovery Systems.

Buyya, R. (2002). Economic-based Distributed Resource Management and Scheduling for Grid Computing. *School of computer science and software engineering Monash University* . Melbourne, Austria.

Buyya, R., Murshed, M. (2002). GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing, Concurrency and Computation: practice and experience (CCPE) wiley press, (1532-0626), 14, 1507-1542.

Caminero, A., Sulistio, A., Caminero, B., Carrion,C., and Buyya, R. (2007). Extending GridSim with an Architecture for failure detection. Parallel and Distributed Systems, International Conference, 1-8.

Cao, G., and Singhal, M. (1999). Mutable Checkpoints: A New Checkpointing Approach for Mobile Computing Systems. *ACM Symposium on Principles of Distributed Computing* .

Crichlow, J. M. (2003). *An Introduction to Distributed and Parallel Computing.* Second edition, PHI.

Dabas, P., and Arya, A. (2013). Grid Computing: An Introduction. *International Journal of Advanced Research in Computer Science and Software Engineering, (2277 128X), 3* .

Devi, D., and Jayanta, Y. S. (2013). A Modified Genetic Algorithm based Load Distribution Approach towards Web Hotspot rescue. *Association of Computer Electronics and Electrical Engineers* .

Ekmecic, I. T., and Milutinovic, V. (1996). A survey of heterogeneous computing: Concepts and Systems. *Proceedings of the IEEE, 84(8)* , 1127-1144.

Elakkiyaa, C. V., Gilead, B. B., and Arvind, A. R. (2014). An Efficient Fault Tolerant Job Scheduling and Load Balancing in Grid Computing. *International Journal of Science and Technoledge* .

Fialho, L. (2011). fault tolerance configuration for uncoordinated checkpoints. *Thesis for the degree of philosophiae doctor* . universitat Antonoma de Barcelona: Bellaterra.

Fidanova, S. (2002). ACO Algorithm with Additional Reinforcement. *Proceeding ANTS '02 Proceedings of the Third International Workshop on Ant Algorithms* , 292-293.

Fidanova, S., and Durchova, M. (2006). Ant algorithm for grid scheduling problem. *In Large-Scale Scientific Computing, Springer Berlin Heidelberg* , 405-412.

Foster, I. (2002). What is the Grid? A Three Point Checklist. Argonne National Laboratory and University of Chicago.

Foster, I., and Kesselman, C. (2002). Computational grids. *In Vector and Parallel Processing—VECPAR 2000* , pp. 3-37. Springer Berlin Heidelberg.

Foster, l. and Kesselman, C. (1999). Computational grids. In *Chapter 2 of "The Grid: Blueprint for a new computing Infrastructure".* Morgan-Kaufman, 1999.

Foster, l., Zhao, Y., Ioan, R., and Shiyong, L. (2008). Cloud computing and grid computing 360-degree compared. *In Grid Computing Environments Workshop GCE'08* , 1-10.

Garg, R., and Kumar, A. S. (2011). Fault Tolerance in Grid computing: State of the art and open issues. *International Journal of Computer Science and Engineering Survey (IJCSES) 2(1)* .

Ghorbannia, A. D., Reza, A., Boroujeni, K., and Bayrampoor, J. (2012). A Balanced Scheduling Algorithm with Fault Tolerant and Task Migration based on Primary Static Mapping (PSM) in Grid. International Journal of Computer Applications 52(8), 10-21.

Gokuldev, S., and Valarmathi, M. (2013). Fault Tolerant System for Computational and Service Grid. *International Journal of Engineering and Innovative Technology (IJEIT) 2(10)* .

Gupta, P. (2011). Grid Computing And Fault Tolerance Approach. *International Journal of Computer Science and Management Studies, 11(3)* , 2231-5268.

http://eu-datagrid.web.cern.ch/eu-datagrid/images/images/grid-small-prov.jpg

Haider, S., Riaz, A. N., Akbar, M., Raza, M. P., and MoyeezUllah, K. G. (2011). Fault Tolerance in Distributed Paradigms. *International Conference on Computer Communication and Management Proc .of CSIT, 5* .

Haque, S., Memon, R. A., and Shaikh, A. (2012). Grid Technology and its Application Developments in Perspective. *International Journal of Independent Research and Studies, (3)* , 112-117.

Hellmuller, R. (2011). Checkpoint/Restart of massively parallel simulations. *Semester thesis* .

Irving, M., Taylor, G., and Hobson, P. (2004). Plug in to grid computing. *Power and Energy Magazine, IEEE* , 2(2), 40-44.

Jacob, B., Brown, M., Fukui, K., and Trivedi, N. (2005). Introduction to Grid Computing. *RedBookInternational Technical Support Organization Introduction to Grid Computing* .

Jafar, S., Krings, A., and Gautier, T. (2009). Flexible Rollback Recovery in Dynamic Heterogeneous Grid Computing. *IEEE Transactions on Dependable and Secure Computing, 6(1)* .

Jayabharathy, J., and Parveen, A. (2009). A Fault Tolerant Load Balancing Model for Grid Environment. *International Journal of Recent Trends in Engineering, 2(2)* .

Kala, K., Rachel, S. I., and Balasubramanian, C. (2014). Pecking Order Continuous Rescue Fault Tolerance Scheduling In Grid Computing. *International Journal of Innovative Research in Science, Engineering and Technology, (3)* .

Keerthika, P., and Kasthuri, N. (2013). A Hybrid Scheduling Algorithm with Load Balancing for Computational Grid. *International Journal of Advanced Science and Technology (58)* , 13-28.

Keerthika, P., and Kasthuri, N. (2011). A New Proactive Fault Tolerant Approach for Scheduling in Computational Grid. *International Conference on Web Services Computing (ICWSC).* Proceedings published by International Journal of Computer Applications® (IJCA).

Kelly, P. (2004). Research Proposal: A Unified Approach to Scheduling in Grid Environments(Doctoral dissertation, The University of Adelaide). http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.90.4751&rep=rep1&type=pdf

Khan, F. G., Qureshi, K., and Nazir, B. (2006). Performance Evolution of Fault Tolerance techniques in Grid Computing System. *Journal of Computers and Electrical Engineering, 36(6)* , 1110 -1122.

Khan, S., and Sharma, N. (2014). Effective Scheduling Algorithm for Load balancing (SALB) using Ant Colony Optimization in Cloud Computing. *International Journal of Advanced Research in Computer Science and Software Engineering 4(2)* .

Khan, S., Hayat, K., Madani, S. A., Khan, S. U., and Kolodziej, J. (2012). The Median Resource failure checkpointing. *COMSATS Institute of Information Technology (CIIT), Abbottabad 22060, Pakistan* , 483-489.

Khanli, L. M., and Analoui, M. (2008). An approach to grid resource selection and fault management based on ECA rules. *Future Generation Computer Systems, 24(4)* , 296-316.

Kumar, R., and Sahoo, G. (2013). Load Balancing Using Ant Colony In Cloud Computing. *International Journal of Information Technology Convergence and Services (IJITCS), 3(5)* .

Kumar, S. G., and Singh, M. (2012). Adaptive and Dynamic Load Balancing in Grid Using Ant Colony Optimization. *International Journal of Engineering and Technology (IJET) 4(4)* .

Lawall, J. L., and Muller, G. (2000). Efficient Incremental Checkpointing of Java Programs. In Dependable Systems and Networks, 2000. DNS 2000. Proceedings International Conference on (pp. 61-70). IEEE.

Li, Y., and Lan, Z. (2006). Exploit Failure Prediction for Adaptive Fault-Tolerance in Cluster. *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06), (1)* , 8.

Liu, Y. Y., Lian, L., and Shi, W. (2006). Using Ant Colony Optimization for SuperScheduling in Computational Grid. *Proceedings of the IEEE Asia-Pacific Conference on Services Computing (APSCC'06)* .

Liu, Y., and Jian, Y. (2006). Performance evaluation and modeling in grid resource management. *Grid and Cooperative Computing. Fifth International Conference* , 335-338.

Lorpunmanee, S., Noor, M. S., Hanan, A. A., and Chompoo, C. I. (2007). An Ant Colony Optimization for Dynamic Job Scheduling in Grid Environment. *International Journal of Computer, Information, Systems and Control Engineering* .

Moallem, A. (2009). Using swarm intelligence for distributed job scheduling on the grid. (Master dissertation, University of Saskatchewan Saskatoon). http://ecommons.usask.ca/.../etd.../thesis.pdf

Mohammad, L. K., Khanli, M. F., and Ghaffari, A. (2010). Reliable Job Scheduler using RFOH in Grid Computing. *Journal of Emerging Trends in Computing and Information Sciences, CIS Journal. 1(1)* .

Mukherjee, A. (2011). An Efficient Job-Grouping Based Scheduling Algorithm for Fine-Grained Jobs in Computational Grids (Doctoral dissertation). http//ethesis.nitrkl.ac.in/2797/1/thesis.pdf

Naik, J. K., Vijaya, K., and Satyanaraya, N. (2012). Scheduling Tasks on Most Suitable Fault tolerant Resource for Execution in Computational Grid. *International Journal of Grid and Distributed Computing 5(3)* .

Nandagopal, M., Basha, C. B., and Ram, N. S. (n.d). Centralized Resource Scheduling with Fault tolerance and Recovery Strategy.

Nandagopal, M., and Uthariaraj, V. R. (2010). Fault Tolerant Scheduling Strategy for Computational Grid Environment. *International Journal of Engineering Science and Technology, 2(9)* , 4361-4372.

Nazir, B., Quresh, K., and Khan, F. G. (2009). Adaptive checkpointing strategy to tolerate faults in economy bapinosed grid. *Journal of Supercomputing* , 1-18.

Nee, D. M., and Gulia, P. (2014). Improve Performance of Load Balancing using Artificial Bee Colony in Grid Computing. *International Journal of Computer Applications (0975 – 8887), 86(14)* .

Ngenkaew, W., Ono, S., and Nakayam, S. (2007). Multiple Pheromone Deposition in Ant-Based Clustering as an Ant Foraging Concept. *Preceedings of the Third IASTED International Conference Advances in Computer Science and Technology.* Phuket, Thailand.

Orbell, K. (2002). *Fault Tolerant Parallel Programming.* Department of Computing Imprial College of Science Technology and Medicine University of London. http://www.orbling.org/ftmpi/report/ftmpirep.pdf

Parvatkar, S., and Gore, D. (2014). An ACO Approach to Maximize Lifetime of Heterogeneous WSN. *(IJCSIT) International Journal of Computer Science and Information Technologies, 5 (4)* .

Patial, S., and Thakur, J. (2014). Checkpointing and Rollback Recovery Algorithms for Fault Tolerance in MANETs. *Int. J. Advanced Networking and Applications 6(3)* .

Pinedo, M. (2005). *Planning and scheduling in manufacturing and services, 24.* New York: Springer.

Pinedo, M. (2002). *Scheduling: theory, algorithms, and systems.* Prentice Hall, second edition.

Poonam, G., and Parveen, K. (2010). Review of Some Check-Pointing Algorithms in Distributed and Mobile Systems. *International Journal of Engineering Science and Technology, 2(6)* , 1594–1602.

Prashar, T., Nancy, and Kumar, D. (2014). Fault Tolerant ACO using Checkpoint in Grid Computing. *International Journal of Computer Applications (0975 – 8887), 98(10)* .

Pritpal, S., and Gurinderpal, S. (2013). Grid Computing Architecture. *International Journal of Computer Science and Communication Engineering* . IJCSCE special issue on "Recent Advances in Engineering and Technology"NCRAET-2013.

Quinn, M. J. (1994). *Parallel computing theory and practice.* McGraw Hill International edition.

Rajeshkumar, J., and Kowsigan, M. (2011). Efficient Scheduling in Computational Grid with an Improved Ant Colony Algorithm. *International Journal of Computer Science and Technology IJCST, 2(4)* .

Ramkumar, B., and Strumpen, V. (1997). Portable checkpointing for heterogeneous architectures, in fault-tolerant computing. *Digest of papers., Twenty-Seventh Annual Internation Symposium Seattle, WA, USA* , 58-57.

Rastogi, R., Nishant, K., and Sharma, P. (2012). Load Balancing of Nodes in Cloud Using Ant Colony Optimization. *Proceedings of the 14th International Conference on Computer Modelling and Simulation (UKSim) IEEE* , 03-08.

Rebbah, M., Slimani, Y., Benye, A., and Brunie, L. (2011). Dynamic Hierarchical Model for Fault Tolerant Grid Computing.

Rhodes, M. N. (2006). A study in grid simulation and scheduling. *Doctoral dissertation* . Durham University: http://etheses.dur.ac.uk./2363/.

Sahu, R. (2011). Many-Objective Comparison of Twelve Grid Scheduling Heuristics. *International Journal, 13(6)* , 9-17.

Sarangi, S. (2009). Particle Swarm Optimisation Applied to Economic Load Dispatch problem. *DEGREE OF Master of Technology In Power Control and Drives* . Department of Electrical Engineering National Institute of Technology Rourkela.

Schopf, J. M. (2002). A General Architecture for Scheduling on the Grid, special issue of JPDC on Grid Computing. http:www.researchgate.net/…/228361579_A_

Singh, B. (2009). Meta-heuristic Techniques for Grid Resource Management. *A thesis Doctor of Philisophy, Computer Science and Engineering Departmet Thapar University, Patiala-147004.*

Srikala, K., and Ramachandram, S. (2013). Fault Tolerant Scheduling of Partitioned and Grouped Jobs in Grid Computing (FTPG). *Global Journal of Computer Science and Technology Cloud and Distributed 13 (2)* .

Subrata, R., and Zomaya, A. Y. (2003). A comparison of three artificial life techniques for reporting cell planning in mobile computing. *IEEE Transactions on Parallel and Distributed Systems* , 142-153.

SunilGavaskar, P., and Subbarao, C. D. (2013). A Survey of Distributed Fault Tolerance Strategies. *International Journal of Advanced Research in Computer and Communication Engineering, 2 (11)* .

Supriya, S., and Dinesh, S. B. (2014). A Survey on Fault Tolerance Mechanisms for job scheduling in Grid computing. *IOSR Journal of Computer Engineering (IOSR-JCE), (2278-0661), 16(2)* , 120-122.

Suryadevera, S., Chourasia, J., Rathore, S., and Jhummarwala, A. (2012). Load Balancing in Computational Grids Using Ant Colony Optimization Algorithm. *International Journal of Computer and Communication Technology (IJCCT)* .

Suryadevera, S., Chourasia, J., Rathore, S., and Jhummarwala, A. (2012). Load Balancing in Computational Grids Using Ant Colony Optimization Algorithm. *International Journal of Computer and Communication Technology (IJCCT), 3(3)* .

Syed, U. A., and Shanmugavel, S. (2013). A Novel Dynamic Reliability Optimized Resource Scheduling Algorithm With Fault Tolerant Approach For Grid Computing System. *Journal of Theoretical and Applied Information Technology, (1992-8645) 589(1)* .

Townend, P., and Xu, J. (2003). Fault tolerance within a grid environment. time-out, 1(S2), S3. http://nesc.ac.uk/events/ahm2003/AHMCD/pdf/063.pdf.

Wolter, K. (2007). Stochastic Models for Restart, Rejuvenation and Checkpointing. *Humboldt-Universitat Berlin Institut fur Informatik Unter den Linden 6, 10099 Berlin, Germany wolter@informatik.hu-berlin.de* .

Wu, K. (2014). A Tunable Workflow Scheduling Algorithm Based On Particle Swarm Optimization For Cloud Computing.http://scholarworks.sjsu.edu/…/vie…

Xue, S., Li, M., Xu, X., and Chen, J. (2014). An ACO-LB Algorithm for Task Scheduling in the Cloud Environment. *JOURNAL OF SOFTWARE, 9(2)* .

Yadav, K., Jindal, D., and Singh, R. (2013). Job Scheduling in Grid Computing. *International Journal of Computer Applications (0975 – 8887), 69(22)* .

Yan, H., Shen, X., Li, X., and Wu, M. (2005). An Improved Ant Algorithm for Job Scheduling in Grid Computing. *In Proceedings of the Fourth International Conference on Machine Learning and Cybernetics* , 2957-2961.

Zhao, G. (2014). Cost-Aware Scheduling Algorithm Based on PSO in Cloud Computing Environment. *International Journal of Grid and Distributed Computing 7(1)* , 33-42.