A NOVEL FAULT TOLERANCE APPROACH TO SCHEDULE JOBS IN GRID ENVIRONMENT

A DISSERTATION

Submitted in partial fulfillment of the requirements for the award of the degree of

MASTER OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

By

VINIT KUMAR

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE -247 667 (INDIA)

JUNE, 2011

CANDIDATE'S DECLARATION

I hereby declare that the work, which is being presented in the dissertation entitled **"A NOVEL FAULT TOLERANCE APPROACH TO SCHEDULE JOBS IN GRID ENVIRONEMNT"** towards the partial fulfillment of the requirement for the award of the degree of **Master of Technology** in **Computer Science and Engineering** submitted to the Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, Roorkee, India is an authentic record of my own work carried out during the period from July 2010 to June 2011, under the guidance of **Dr. Padam Kumar, Professor, Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee.**

The matter presented in this dissertation has not been submitted by me for the award of any other degree of this or any other Institute.

Date: 01/06/2011
Place: Roorkee.

Vinit.
(VINIT KUMAR)

CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of my knowledge and belief.

Date: 1/6/11
Place: Roorkee.

(Dr. PADAM KUMAR)
Professor
Department of Electronics and Computer Engineering
Indian Institute of Technology Roorkee

ACKNOWLEDGEMENT

It gives me immense pleasure to express my deepest sense of gratitude towards my guide **Dr. Padam Kumar,** Professor, Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee for his expert guidance, encouragement and support throughout the dissertation work. His suggestions and invaluable ideas provided the platform to the entire dissertation work. In spite of his extremely busy schedule, I have always found him accessible for suggestions and discussions. I look at him with great respect for his profound knowledge and relentless pursuit for perfection. His ever-encouraging attitude and help has been immensely valuable.

Nothing would have been possible without the support of my family members, who have been backing me up throughout my life. I wish to convey my sincere thanks to my parents. Without their support, it would not be possible to reach this far with my studies.

I would also like to thank all faculty members of Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee for their kind help and support.

VINIT KUMAR

ABSTRACT

Grids are type of distributed machines in which aggregation of resources is done to provide services. They are present as systems which extend internet environments with machines distributed across multiple organizations and administrative domains. The power of Grid computing depends upon abundance of network connected systems and bandwidth available for computation, collaboration and communication over the network. Dependable, consistent, pervasive and inexpensive computing infrastructure is the core of Grid computing. Its virtual image provides a single point of access to powerful distributed resources. It enables the deployment of resources such as desktops, databases and storages wherever and whenever needed, as has been demonstrated by the projects such as SETI@home, MyGrid. As more and more resources are connecting over network, the grid size becomes more large, which increase the probability of failure due to network isolation, resource failure etc. To achieve high performance in grid, task must be continue its processing in the presence of faults, which means to provide a fault tolerance environment to task, so that high performance can be achieved in grid.

In this dissertation, a novel fault tolerance approach is proposed. Fault tolerance approaches mainly lie into two categories, Replica Based approach and Check-Pointing approach. The main problem with Replica Based approach is its non-applicability to cost based resources whereas Check-Pointing approach suffers from inherent disadvantages of taking Check-Point, like overhead and wastage of time. Till now no one has focused on nature of tasks submitted to grid. In this dissertation, a novel approach for fault tolerance is proposed. In this approach, large task is divided into various subtasks on the basis of data flow and control flow dependencies. The experimental results show that proposed approach is efficient than Check-Point approach in terms of various parameters like number of Gridlets successfully completed and average execution time of task.

TABLE OF CONTENTS

Candidate's Declaration and Certificate.....i

Acknowledgement.....ii

Abstract.....iii

Table of Contents.....iv

List of Figures.....vi

List of Tables.....viii

1. Introduction and Statement of the Problem.....**1**

1.1 Introduction.....1

1.2 Motivation.....2

1.3 Statement of the Problem.....3

1.4 Organization of the Report.....3

2. Background and Literature Review.....**4**

2.1 Grid : An Overview.....4

2.1.1 Types of Grid.....5

2.2 Faults in Grid.....6

2.2.1 The Survey.....7

2.2.2 Survey Conclusion.....9

2.3 Literature Review.....10

2.3.1 Replica Based Approach.....10

2.3.2 Check-Point Approach.....18

3. Resubmission Based Approach.....**26**

3.1 Working Principle.....26

3.2 System Model.....27

3.3 Algorithms.....30

3.4 Salient Features.....33

4. Simulation Tool and Parameters 34

4.1 GridSim: Grid Modeling and Simulation Toolkit 34

4.2 GridSim Entites 35

4.3 Application Model 37

4.4 Resource Model 37

4.5 Simulation Environment and Data 38

5. Results and Discussions.....40

5.1 Experiment Results..... 40

5.2 Comparison of Check-Pointing approach and Resubmission Based approach 42

5.2.1 Different values of budget 42

5.2.2 Different values of deadline time 44

6. Conclusions and Scope for Future Work.....46

6.1 Conclusions.....46

6.2 Scope for Future Work.....46

REFERENCES.....47

LIST OF PUBLICATIONS.....49

LIST OF FIGURES

Figure 2.1	Grid Systems	6
Figure 2.2	Kinds of Failure	8
Figure 2.3	Fault Treatment Mechanisms in Current Use	8
Figure 2.4	Greatest User Complaints	9
Figure 2.5	Degree of User Involvement	9
Figure 2.6	The Replication Model for the Grid	12
Figure 2.7	Operation of the STA Algorithm	16
Figure 2.8	Five level of security demand	17
Figure 2.9	Five level of replica	17
Figure 2.10	Flow Chart of calculation of Number of optimal replica	18
Figure 2.11	Process state (shaded area) vs. process environment (non-shaded area)	20
Figure 2.12	Architecture of low level check-pointing packages	22
Figure 3.1	Resource life time	26
Figure 3.2	System Architecture	27
Figure 3.3	Interaction of different components	29
Figure 4.1	Flow diagram in GridSim based Simulation	35
Figure 5.1	Precedence Constraints Directed Acyclic Flow Graph	41
Figure 5.2	Number of job success for different values of budget	43

Figure 5.3 Number of jobs fail to achieve deadline..... 44

Figure 5.4 Number of job success for different values of deadline..... 45

LIST OF TABLES

Table 3.1	Relationship between Job Status and Resource Life Time	26
Table 4.1	Simulation Parameters.....	38

Chapter 1

Introduction and Statement of the Problem

1.1 Introduction

Grid is "*A type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed autonomous and heterogeneous resources dynamically at runtime depending on their availability, capability, performance, cost and users' quality-of-service requirements*" [1]. Grid computing [2] enables aggregation and sharing of geographically distributed resources and data into a single virtual machine for solving the large-scale problems, which requires more computational power. Grid can be used for many applications like medical imaging and diagnosis, biometrics, satellite image processing. Grid has many design issues like scheduling, data management, resource management, security, load balancing and fault tolerance. Grid jobs are very large and our grid environment is more likely to have failure, so fault management becomes more important to give desired Quality of Service (QoS) to grid user. Due to large size of jobs, it may also be the case that the cost and difficulty of finding and recovering from faults in Grid applications is higher than normal applications. In grid environment resources may enter and leave at any time which may cause fault. The term fault tolerance means to continue the work correctly in the presence of fault.

Research on fault tolerance in the grid environment divides its mechanism into two main types: proactive and post-active. Proactive fault tolerance mechanism takes into account the failure of grid resource before scheduling jobs on grid resources, like replication approach. On the other hand, the post-active mechanism takes appropriate action after the job failure, like check pointing approach.

Fault Tolerance approach requires three main steps, failure detection, failure notification and failure recovery. Failure detection is the phase between the failure occurrence and the time in which the failure is discovered. Failure notification is the phase between the failure detection and the instant in which nodes responsible for

recovering the failure are notified. Failure recovery is the phase between failure notification and the time in which the pre-failure working conditions are recovered.

1.2 Motivation

The performance of Grid is effected by various issues such as scheduling, load balancing, fault tolerance, resource discovery etc. Fault tolerance is one of the important issues in Grid computing. Number of failures in a Grid depends upon two factors: size of grid and size of jobs. As more numbers of resources are connected to the Grid, there are more chances of failure in terms of node isolation, network disconnection, power failure at nodes. Also to take advantages from the Grid's computational power, applications' size increase tremendously. As the size of grid jobs increase, job will be for longer duration in the Grid. This will increase the chances of failure due to vulnerable environment inside Grid. In that case, failure of a resource may lead to restart all large tasks which are currently being executed on that resource. This leads to heavy economic and computational losses. To overcome these problems, fault tolerance is provided in the grid environment. In the literature, there are two basic approaches used, Replica Based approach and Check-Point approach. Replica Based approach uses the principle of probability; it means executing multiple copies of a task increases the probability of successful completion of task. As the name shows, Check-Point approach uses the principle of Check-Pointing, means taking Check-Point of task at regular intervals and after failure restart the task from last stored Check-point.

In this report, a novel fault tolerance approach is proposed, which uses the principle of dividing the large task into small sub tasks, and schedules them on different resources on the basis of different dependencies.

1.3 Statement of the Problem

The main aim of this dissertation is to design and implement a novel fault tolerance approach for scheduling jobs in Grid environment. The problem is addressed by using following steps:

- Study the grid technologies in the areas concerned with fault tolerance environment in grid.
- Propose a fault tolerance approach for grid environment to guarantee continuous and reliable execution of task in spite of resource failure.
- Evaluate the performance of proposed approach, taking different resource capability, deadline and budget.

1.4 Organization of the Report

Rest of the report is organized in various chapters including this one which introduces the topic and states the problem.

Chapter 2 gives an overview of grid computing and faults in grid.

Chapter 3 presents brief description about work done in field of fault tolerance in grid computing.

Chapter 4 describes the details of Resubmission Based Fault Tolerance approach.

Chapter 5 discusses about simulation experiment setup, description of classes used and GridSim-the simulator used.

Chapter 6 discusses simulation results obtained under various conditions i.e. different resource capability, deadline and budget.

Chapter 7 concludes the dissertation and provides directions for the future work.

Chapter 2

Background and Literature Review

2.1 Grid: An Overview

Grids are the type of distributed systems in which aggregation of resources is done to provide services. They are present as systems which spread internet environments with machines distributed across multiple organizations and administrative domains. Grid computing [2] works on the potential in the growth and abundance of network connected systems and bandwidth: computation, collaboration and communication over the network. At the core of grid computing is a computing infrastructure that provides dependable, consistent, pervasive and inexpensive access to computational capabilities. By aggregating large assets into a virtual system, a grid provides a single point of access to powerful distributed resources. With a grid, networked resources such as desktops, servers, storage, databases, and even scientific instruments can be combined to deploy massive computing power wherever and whenever it is needed most. The term 'Grid' is used to describe a number of different, but related, ideas, including utility computing concepts, grid technologies, and grid standards. In recent times the term 'Grid' is used in the widest sense to describe the ability to aggregate and share Information Technology (IT) resources in a global environment in a manner which achieves seamless, secure, transparent, simple access to a large collection of many different types of hardware and software resources, (including computer nodes, software codes, data repositories, storage devices, graphics and terminal devices and instrumentation and equipment), through non-dedicated wide area networks, to deliver customized resources to specific applications.

A dynamic set of individuals and/or institutions defined around a set of resource-sharing rules and conditions, these set of individuals corresponds to the virtual organization (VO). All VOs share some commonality with respect to architecture, platform, or some miscellaneous characteristics.

Grid architecture identifies fundamental system components, specifies the purpose and function of these components, and indicates how these components interact with one another. The goal is not to provide a complete enumeration of all required

components but to identify requirements for general component classes. The result is an extensible, open architectural structure within which can be placed solutions to key VO requirements.

Actual computational grid can span several management domains, and in these domains the very high trust relationship must be supported each other. This is because sometime for supporting powerful computing, computational resources in different domains usually should be shared; while this kind of resource sharing may lead to illegal users acquire much higher security level to access to the resources that they have no rights to access to. Under these conditions, the security of Grid may be threatened. In addition to this, if illegal users run malicious code in the environment of computational Grid, certain resources in it may be destroyed and all information in it will disappear forever. For addressing these problems trust plays a vital role in grid environment.

2.1.1 Types of Grid

Grid computing can be used in variety of ways to address various kinds of application requirements. According to the distinct application realms, grid systems can be classified into three categories but there are actually no hard boundaries between these grid categories. Real grids may be a combination of one or more of these types. The three categories [3] of grid systems are describe as below:

- **Computational Grid**

A computational grid system that aims at achieving higher aggregate computation power than any single constitute machine. According to how the computing power is utilized, computational grids can be further subdivided into *distributed supercomputing* and *high throughput* categories. A distributed supercomputing grid exploits the parallel execution of applications over multiple machines simultaneously to reduce the execution time. A high throughput grid aims to increase the completion rate of a stream of jobs through utilizing available idle computing cycles as many as possible.

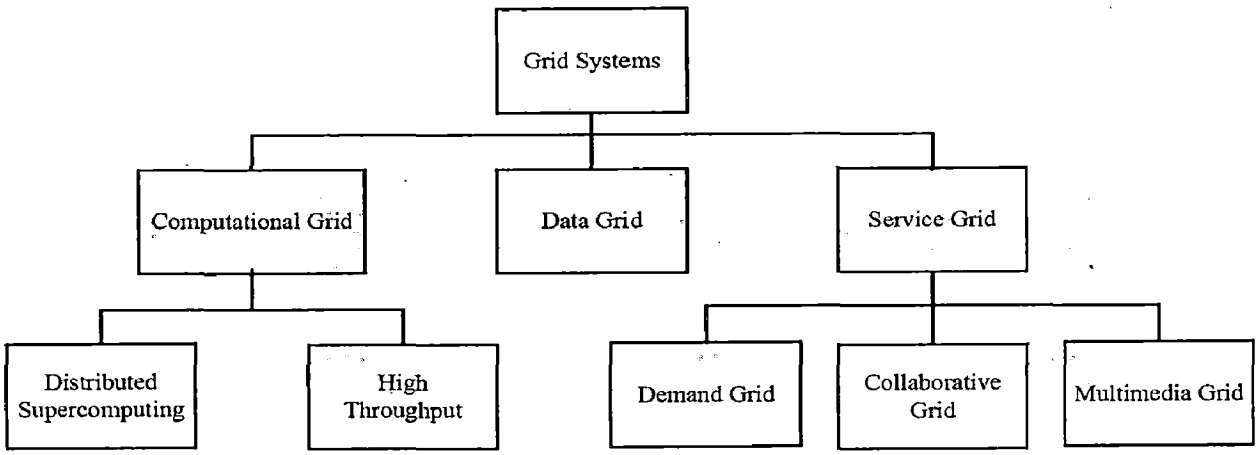


Figure 2.1 Grid Systems

- **Data Grid**

Data grid is mainly used for providing access to the data which spreads across various organizations. Users only have to care about the location of data until they have access to that data. For example data grid allows two organizations doing research on cloud computing having unique data to share their data and other processing on that data.

- **Service Grid**

A service grid provides the services those are not provided by any single machine. This type of grid is further classified in demand, collaborative and multimedia grid. Demand grid dynamically aggregates different resources to provide new services. Collaborative grid provides real time interaction between users and applications via a virtual workspace by connecting users and applications into collaborative environment. A multimedia grid provides an infrastructure for real time multimedia applications.

2.2 Faults in Grid

A grid environment is made up of thousands of resources, application and services which need to interact with each other to provide a powerful executing platform. Since these elements are extremely heterogeneous, grids are likely to failure. These failures are not only individual failure; they may cause multiple sites failure, where

multiple sites are interacting. Moreover machines may be disconnected from the grid due to machine failures, network partitions or due to many other reasons. So we conclude that grid is more prone to failure than traditional computing platform. So dealing with failure in such environment is a big challenge. Detecting the fault is not a big deal, the main difficulty arises in finding out the root of the problem i.e. to diagnose a failure is a complex task.

2.2.1 The Survey

To correlate such discussion with real world a survey [4] has been done. In the survey, some multiple choice questions are asked to the grid users and user can select more than one option, that's why in some graph summation of all options may be larger than 100%.

- **Kinds of failure**

The main kinds of failures (Figure 2.2) are related to the environment configuration. Almost 76% of the responses have pointed this out. Survey reveals the fact that the main cause of configuration failures in Grid is the lack of control over grid resources. Also survey shows different types of failures like middleware failures (48%), application failures (43%) and hardware failures (34%).

- **Fault Treatment Mechanisms**

Figure 2.3 shows various fault treatment mechanisms used by Grid users. There are some ad-hoc mechanisms which are based on complaints of users' and analysis of log files. There are also some automated ways to deal with these failures. Some of them are dependent on application. Some of them are proprietary ones even in the presence of monitoring systems. Check-pointing and fault-tolerant scheduling are other solutions to treat failures in grid environment. These two techniques are able to deal with crash failure for hardware and software.

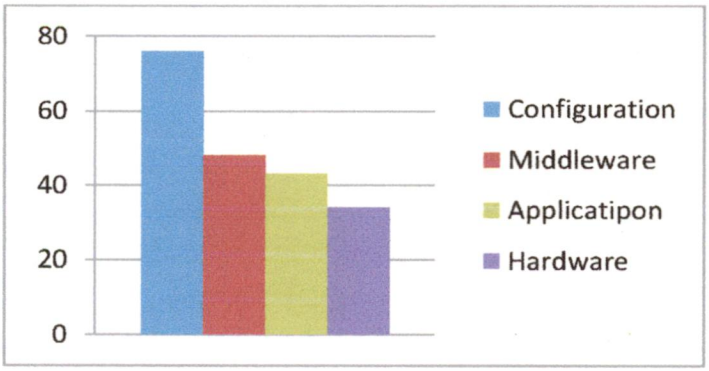


Figure 2.2 Kinds of Failure

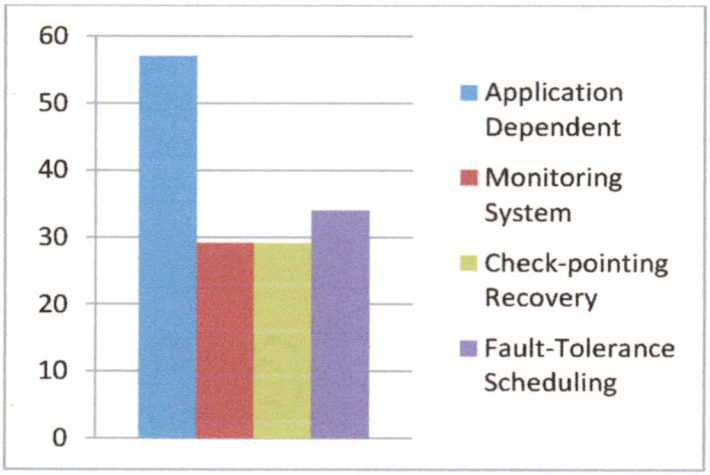


Figure 2.3 Fault Treatment Mechanisms in Current Use

- **Greatest Problems for Recovering from a Failure**

Figure 2.4 shows the problems faced by Grid users in case of a failure. As the figure shows the most common problem is identifying the main cause of failure. Some says the difficulty faced in implementing application dependent failure recovery. It shows the inability in recovery from a failure. Other problems include orphaned jobs on other systems which affect the performance of the system, corrupted cache files, unable to access to preserved state in case of check-pointing technique is used.

- **Degree of User Involvement**

Figure 2.5 shows the requirement of user involvement in case of failure recovery process. It involves things to be done when failures occur or notification of user when serious errors happen. Survey shows that user involvement in failure recovery process is very low and they are dependent on the mechanisms of systems.

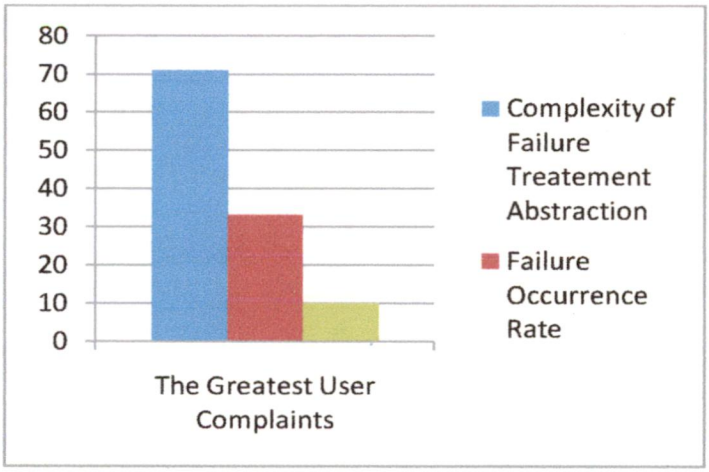


Figure 2.4 Greatest User Complaints

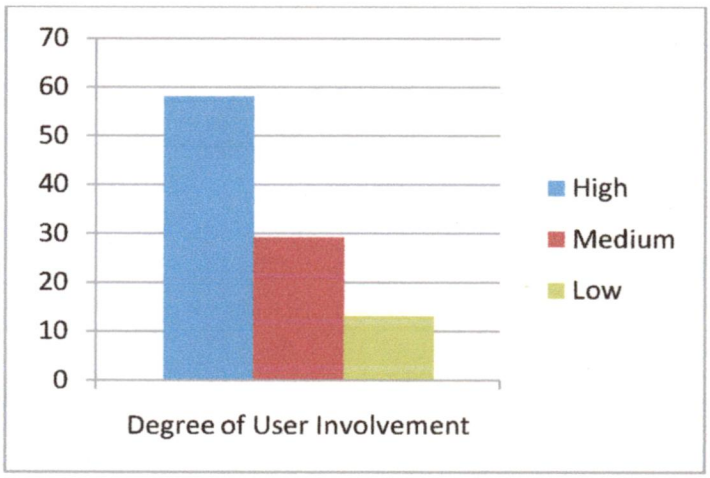


Figure 2.5 Degree of User Involvement

2.2.2 Survey Conclusion

From the responses above, we can infer that failures are not rare. The main source of failures is related to configuration issues and failure diagnosis is the main problem. This scenario is a result of the following fact:

Grid is made up of various components like middleware, resource broker etc. One component uses facilities provided by other component. For example, an application uses facilities provided by middleware which in turn uses the abstractions of operating systems. Hence whenever there is a problem in one component, it propagates it in other components also. Hence one has to narrow down to different components to find out the original source of the failure. This is exactly like the backtracking technique used by programming languages to find the source of error. The problem is that, when everything works, one has to know only *what* a software component does, but when things break, one has also to know *how* the component works. Although not exclusive

of grids, this characterization is a much bigger problem in grids than in traditional systems. This is because grids are much more complex and heterogeneous, encompassing a much greater number of technologies than traditional computing systems. In a grid, one can discover a failure in a grid processor about what user could never know it's hardware platform model has existed. Thus user knows nothing about it, how it should work, where its logs are. Thus, solving the problem is a very difficult task. Hence there is an obstruction between detection and diagnosis of failure. Actually log files are available to diagnose the problem but the user who uses them is not able to interpret them. At this point there is a great requirement of user association. But at this point focus of developer is lost as he has to focus on functionality as well.

2.3 Literature Review

2.3.1 Replica-Based Approach

Task replication [5] is an approach used in grid to provide fault tolerance by scheduling multiple copies of same task, so that to increase the probability of success of at least one copy. The basic idea is to produces multiple instance of a given task, based on the fact that failure probability is known a priori. In such approach, if a replica fails then it does not restarted and success of task is mainly depends on the success of other replicas. The replica approach causes an overhead in workload due to executing multiple copies instead of single. If there are 'n' identical copies for a task of workload 'w', then the total workload of grid is given by $w \times n$. Number of replica has a linear relationship with workload on the grid infrastructure. But due to its good performance replica approach is in use at many grid solutions.

Let there is a set of M processors, which form a grid infrastructure. Each processor has a fixed computational capacity denoted as Cj, where j $\in \{1, 2, 3, \ldots\ldots, M\}$, thus the total computational capacity of grid is $C = \sum_{j=1}^{M} Cj$. We also consider a set of N different tasks T_i, $i \in \{1, 2, 3 \ldots N\}$.

Lets the failure probability of a task T_i *is* Pf_i, which is the probability of failure of the task executed on grid. Similarly success probability Ps_i *is* the probability of success of the task T_i in executed on grid. There is a correlation between them as follow

$$Pf_i = 1 - Ps_i \qquad \text{....eqn(2.1)}$$

Task replicas are generated and assigned to grid infrastructure for execution. By this, a low probability of task failure can be achieved. Lets k replicas of a task T_i *is* produced, denoted by T_{i1}, to T_{ik}. Similarly the failure probabilities are given by Pf_{i1} to Pf_{ik} and the failure probability of task T_i is given by following equation.

$$Pf_i = \prod_{j=1}^{k} Pij \qquad \text{....eqn(2.2)}$$

Similarly probability of success of task T_i is give by:

$$Ps_i = 1 - Pf_i = 1 - \prod_{j=1}^{k} pfij \qquad \text{....eqn(2.3)}$$

The value of k is decided by the probability threshold δ, means in order to provide a low failure probability we produce as many task replicas as need to satisfy the constraint of success probability δ. We can write as:

$$Pf_i \leq \delta \qquad \text{....eqn(2.4)}$$

Where δ is a constant between 0 and 1.

- **A Group Communicator for Grid**

Communication between replicas is an important part of replication based fault tolerance. Replicas multicast the messages for other replicas and other replicas must receive these messages in same order so that all replicas must be in identical state. To achieve this, Total Order Multicast protocol (TOM) [6] is implemented, which ensures that all replicas receive the multicast messages in the same order. To achieve TOM, grid is partitioned into clusters. Each cluster is controlled by a cluster head called as coordinator. These coordinators are interfaces for the internal node of a cluster to the external node in other clusters in network. They perform TOM on the behalf of an ordinary node. We create groups of process for communication. Group membership services are used to manage a group of processes and are based on the view, which is the list of processes belonging to a group at a particular time. Change in view notified to all other members. The basic operations provided are join, leave, exclude. When a process p want to join a group it uses join operation and all other processes are notified for this action. A process can be removed from a group by using exclusion operation, if its crash is detected by a member of group. An exit is a voluntarily release of process from a group by itself. There are two basic primitives;

send_multicast to send a multicast message and receive_multicast to receive a message, sent by a process which is a member of group. Atomic broadcast is a special case of total multicast where TOM message is delivered to all of group or none.

System Model

For implementing TOM, we have to divide Grid in clusters. Figure 2.6 shows the architecture where different replicas are combined to form a cluster which is represented by a coordinator. Then clusters are combined to form a process group which is called replica group. A coordinator group is formed by the coordinators of different clusters. Whenever previous coordinator crashes, a new coordinator is given. Coordinator uses multicast communication to communicate in its groups.

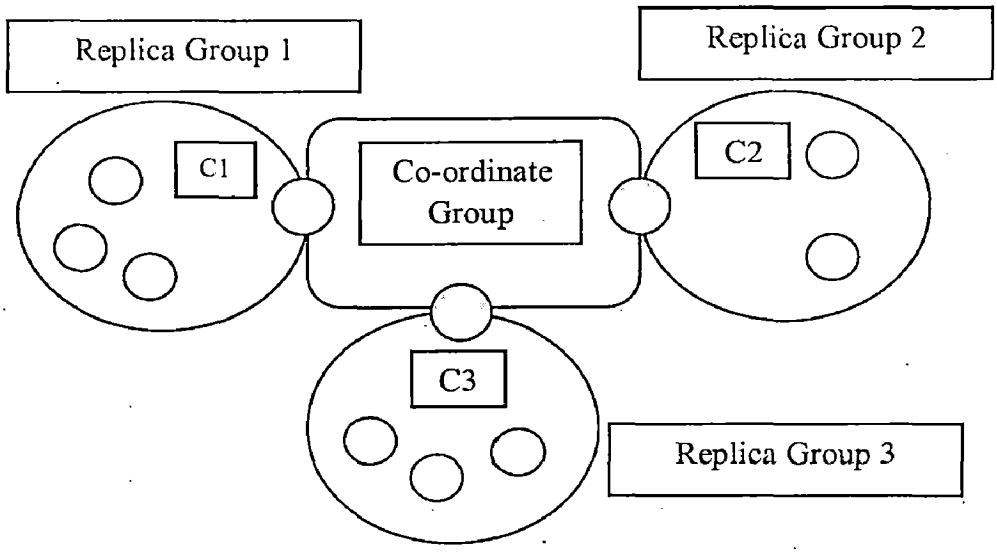


Figure 2.6 The Replication Model for the Grid

TOM Protocol

The Tom protocol used here is Static Token Algorithm which employs similar data structure for the token as in Suzuki-Kasam [7] algorithm for distributed mutual exclusion. A process that has the system wide token has the right to send a TOM message. The token data structure is as follow:

- Token_sequence_Number(TSN) : integer;
- Token_Request_Queue(TRQ) : Queue of nodes;

Each node uses a sequence no. which is local which is actually the seq. no. of last message. Whenever a node has to send a message, it has to send a request to

coordinator. The coordinator of that group (node's group) spreads that message to its fellow coordinators present in coordinator group. The coordinator set its state to waiting state and waits for token. When coordinator receives the token, it changes its state to hold token state and forward this token to the requesting node. Requesting node sends TOM_Msg along with token and coordinator multicasts the message and sends the token to requesting coordinator. After that coordinator sends a TOM_Chk message to check the acknowledgement, if all the nodes in group have received the message, the operation is successful and a final TOM_Set message is sent to all coordinator to allow the final delivery to nodes. The node which receives TOM_Set message checks that if this message has arrived in sequence or not. If yes, it is transmitted to application otherwise delayed.

Illustration of STA

Figure 2.7 shows an example scenario for STA. Initially, Token is held by coordinator C_2. For simplicity, it is assumed that coordinators are directly connected with the nodes and each others. Following is the sequence of events:

1. Node $n13$ in cluster 1 requests Token from its coordinator $C1$.

2. $C1$, does not have the Token, hence broadcasts this request in the coordinator group.

3. $C2$ has the token which is not being used and has TSN as 0 and its queue is empty. $C2$ sets the destination of the token as $n13$ and sends this token to $C1$.

4. $C1$ receives the Token and sends it to $n13$ which is received by it. These first four steps are depicted in Figure 2.7(a).

5. While $n13$ is holding the Token, nodes $n21$ and $n32$ in clusters 2 and 3 make requests consecutively to their coordinators for the Token which in turn send requests $R21$ and $R32$ to the coordinator group.

6. $R21$ and then $R32$ reach $C1$ consecutively. $C1$ queues these requests.

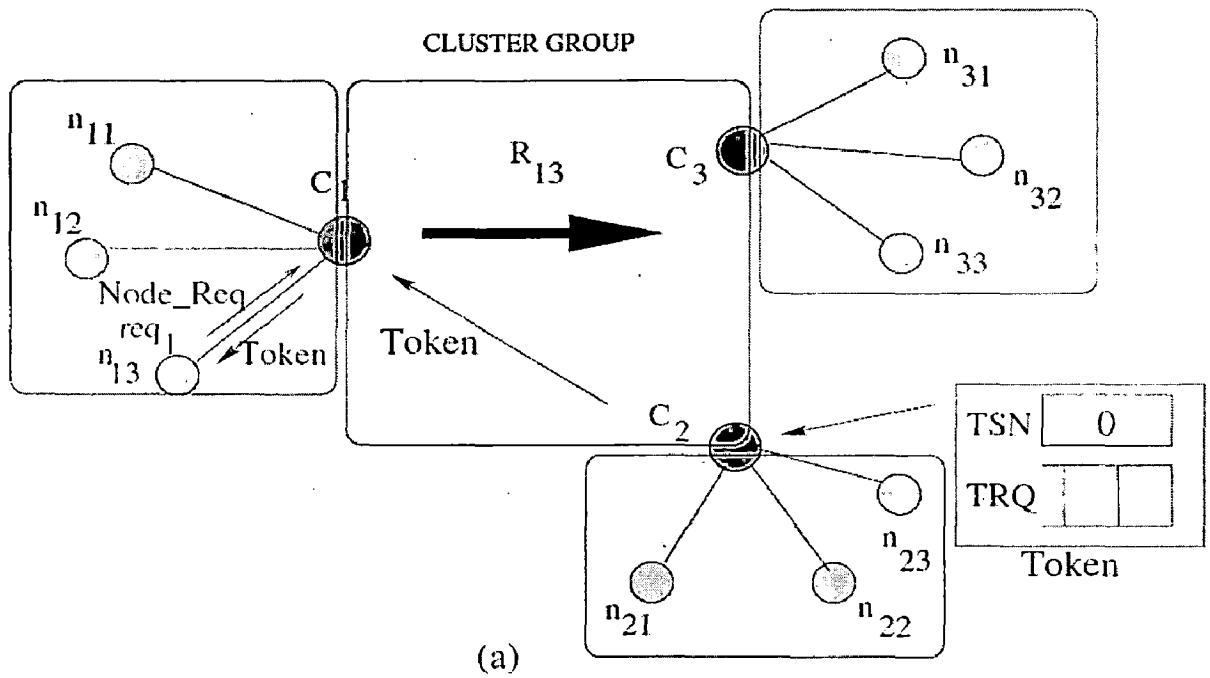
7. When $n13$ receives the token, it increments TSN of the Token to 1 and sends TOM message with this sequence number to $C1$ along with the Token. Steps 5-6-7 are depicted in Figure 2.7(b).

8. *C*1 receives the Token and the TOM message. It broadcasts TOM on the coordinator group. *C*1 also checks its local Coordinator Token Request Queue (CRQ). It appends the nodes in its local queue to the Token Queue, removes the first node from the TRQ ($n31$) and sends the token to *C*2.

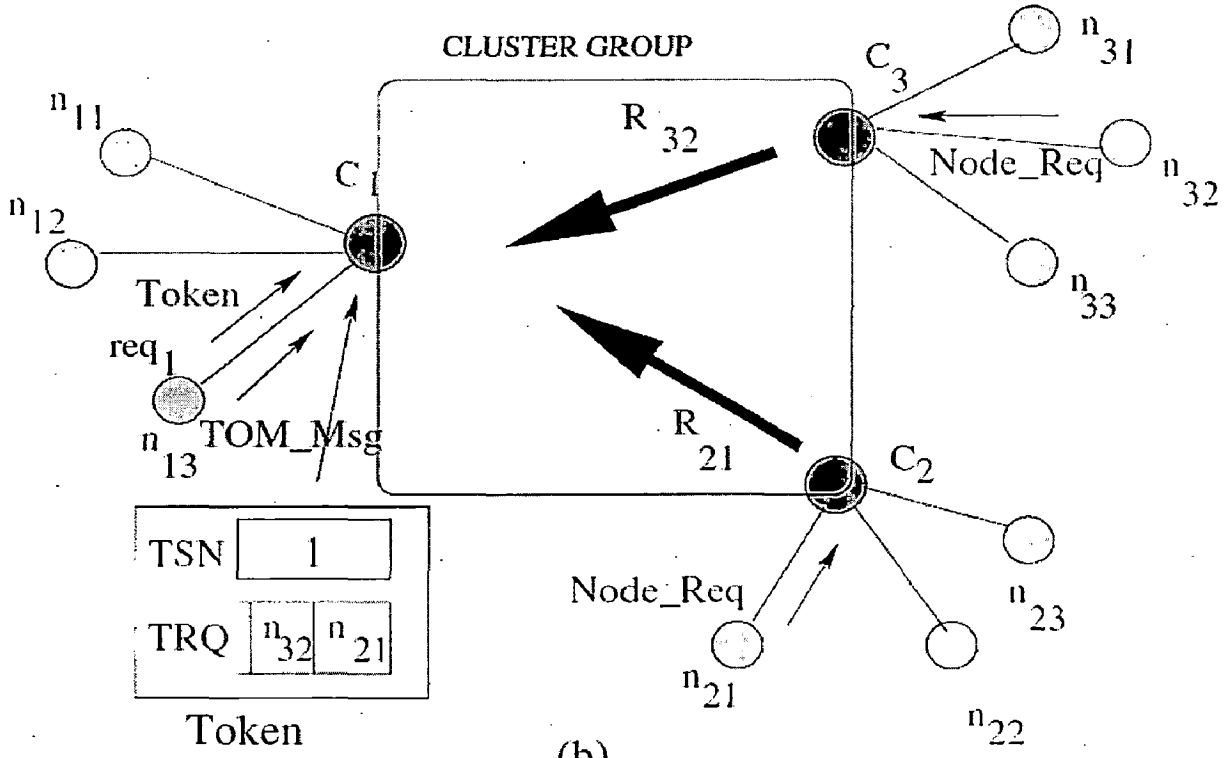
9. *C*3 and *C*2, pass an acknowledgement message (*TOM Ack*) to the source. Steps 8 and 9 are depicted in Figure 2.7(c).

10. If *C*1 receives (*TOM Ack*) acknowledgements from every node in the group, the TOM is successful. In this case, *C*1 issues a *TOM Set* message to finally initiate the actual delivery of the TOM message to the application. The replica node however, checks its LSN with TSN to conclude TOM delivery as described above.

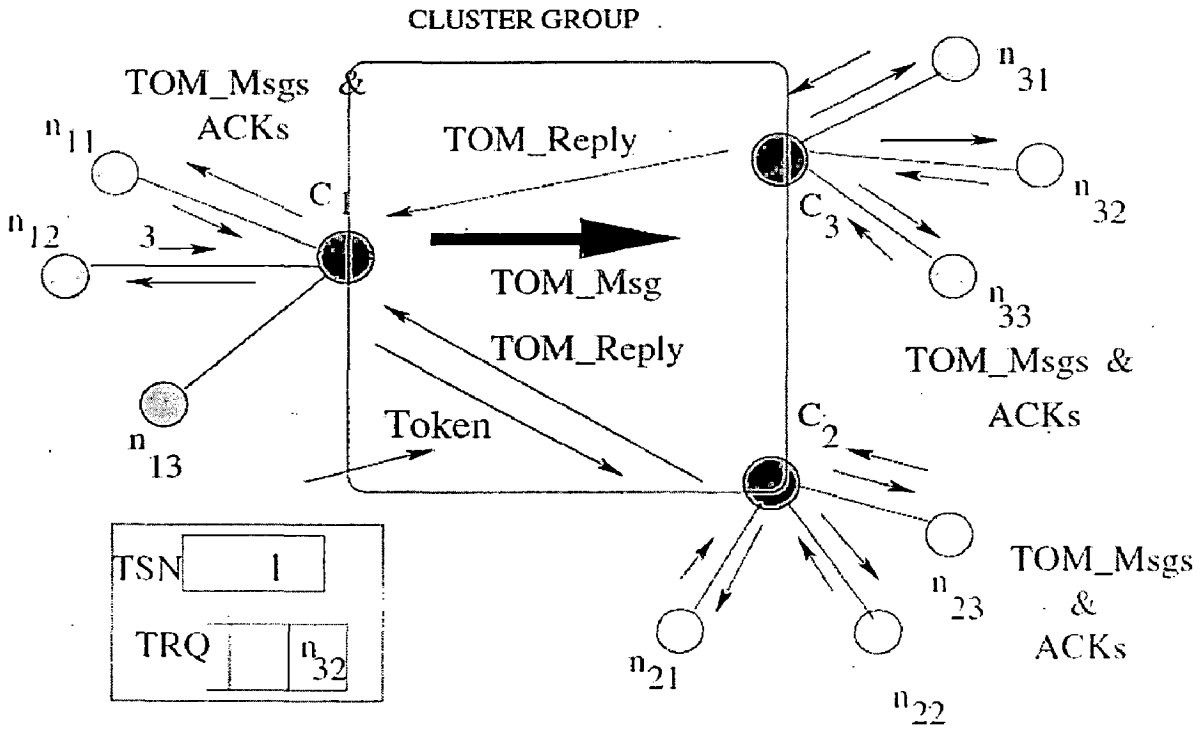
11. When *C*2 receives the Token from *C*1, it proceeds similar to 7-8-9-10 above and when $n21$ finishes with the Token, *C*2 sends it to *C*3. Steps 10 and 11 are depicted in Figure 2.7(d). Note that this is performed in parallel with the *TOM Msg* delivery of $n13$.



(a)


CLUSTER GROUP
C3
n31
R32
n11
C1
Node_Req
n32
n12
Token
n33
req1
n13
TOM_Msg
R21
C2
TSN
1
TRQ
n32
n21
Node_Req
n23
n21
n22
Token

(b)


CLUSTER GROUP
TOM_Msgs &
ACKs
n31
n11
TOM_Reply
C1
n32
n12
3
C3
TOM_Msg
n33
TOM_Reply
TOM_Msgs &
ACKs
Token
C2
n13
TSN
1
TRQ
n32
TOM_Msgs
&
ACKs
n23
n21
n22

(c)

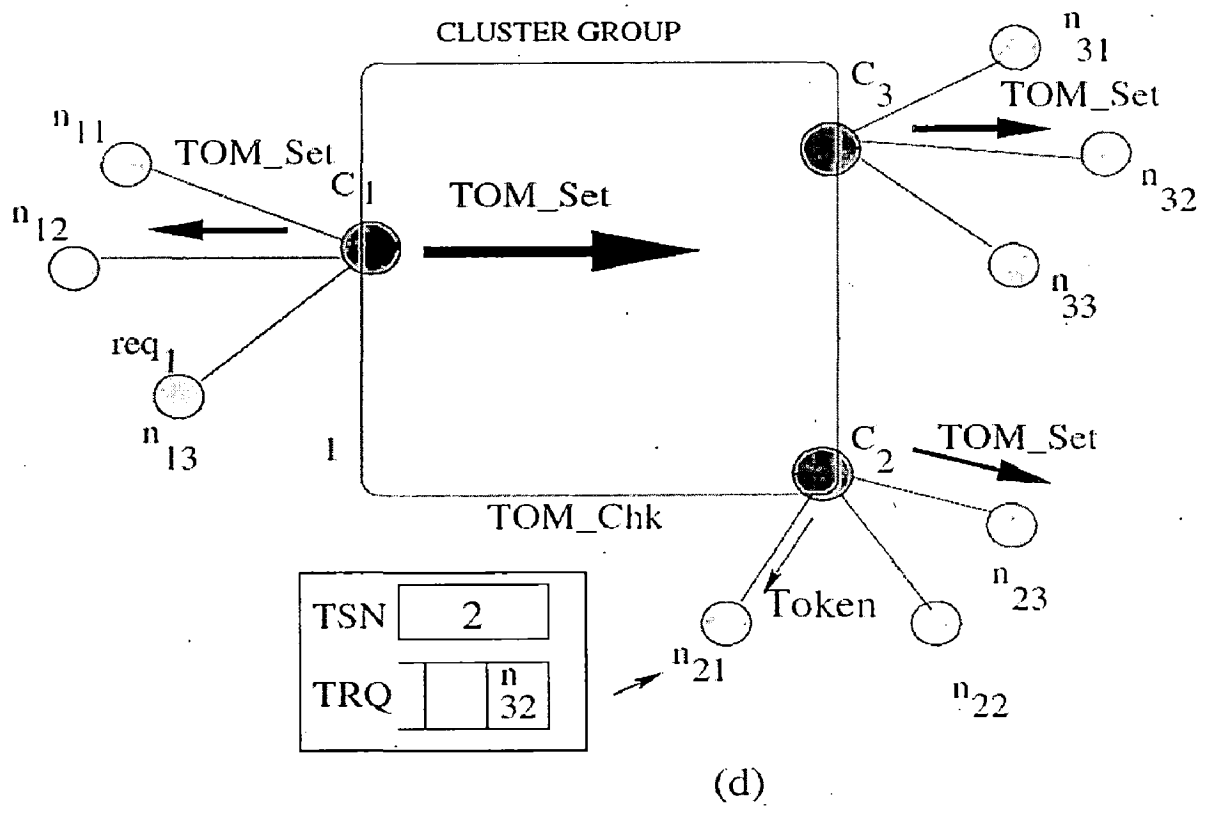


Figure 2.7 Operation of the STA Algorithm

- **A Fuzzy Approach to Calculate Number of Replica**

As explained in previous section number of replica used, has a relation with workload of grid infrastructure. However, existing job replication algorithms use a fixed number of replication and increase the workload whenever a new job arrives for execution. Although the grid environment is dynamic in nature (work load, security demand, trust level etc.), so an adaptive approach [7] can be used for deciding the number of replicas require to give the optimal result at any given time depending on the current grid environment. The security level of grid is dynamic in nature, because there is no way to predict when and where a grid will be under attack or crash. Similarly security demand of an application is also changing with time due to budget or deadline or any other reason which affects user policy. Now mainly two parameters are under consideration; one is security demand (SD) of application which is taken during job submission time from the user, and second is trust level which is the trustworthiness of grid environment, managed throughout the life cycle of grid. The dynamic nature of these parameters can be presented by two fuzzy set. Depending on these two fuzzy set we can calculate the number of replicas require for grid at any given time using rule base fuzzy logic.

The Fuzzy Inference Process

A fuzzy set expresses the degree to which an element belongs to a set, this is called membership value. The membership value of an element in a fuzzy set is calculated with the help of membership function. The membership function assigns each element a value between 0 and 1. Where 0 represent no containment and 1 represents full containment. Two fuzzy set are shown below: the security demand and the number of replica required.

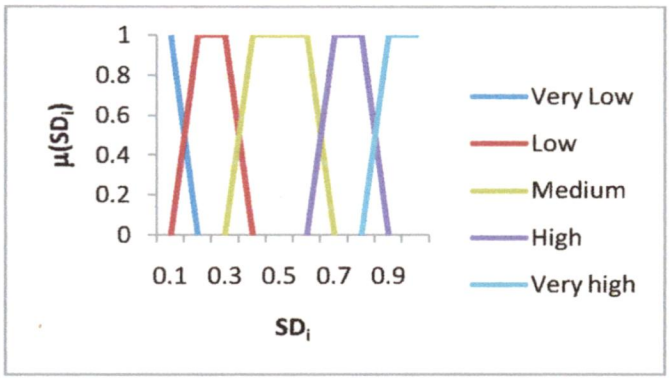


Figure 2.8 Five level of security demand

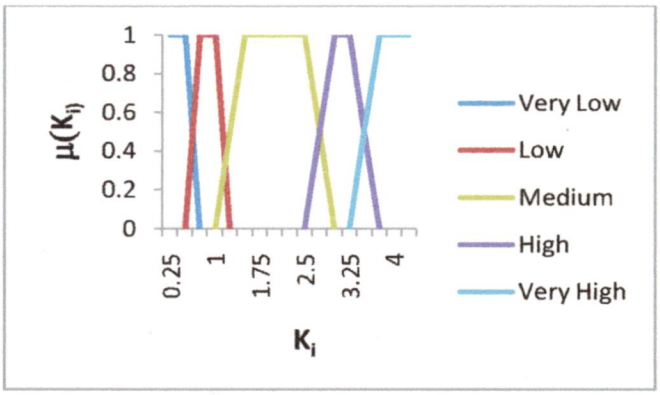


Figure 2.9 Five level of replica

Flow Chart:

Figure 2.10 shows the flow chart of calculation of Number of optimal replica.

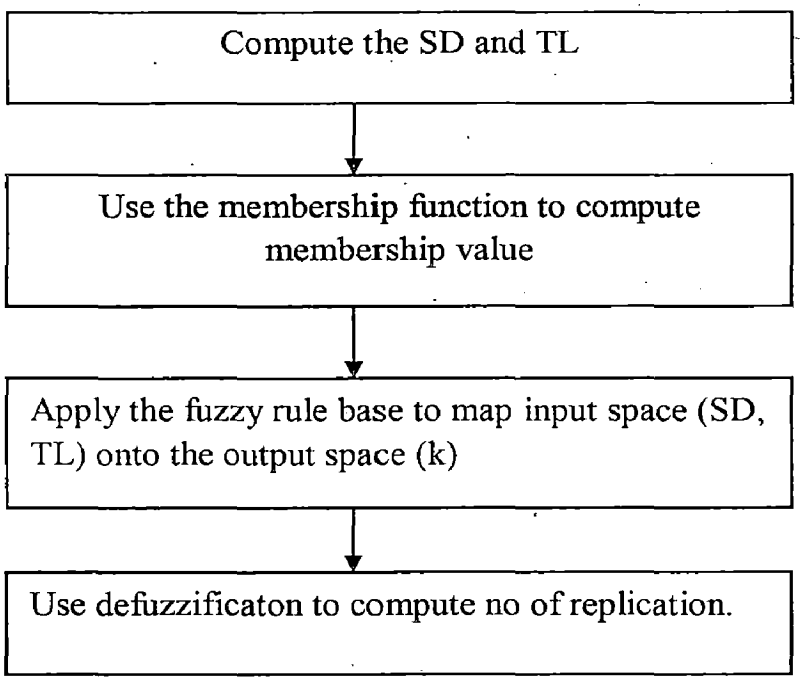


Figure 2.10 Flow Chart of calculation of Number of optimal replica

Some rule bases are used in fuzzy inference, which are constructed with the past experience. These rule bases are use for taking the decision. Example of such rules is following:

Rule 1: IF SD is medium and TL high is THEN k is medium

Rule 2: IF SD is low and TL high is THEN k is low

2.3.2 Check-Pointing Approach

Check-pointing [9] means, we save the running state of the process into an image file, and then restarts the program based on the image file. So the basic questions are

1) Which state need to be saved

2) How to save these states

3) How to restore the states

Handling persistent state of process in HPC environment [10] is one of the solutions. The state of a program in execution has many aspects. Basically, all the process private data needs to be saved and recovered during check-pointing. This includes: address space, register set, open files/pipes/sockets, System V IPC structures, current working directory, signal handlers, timers, terminal settings, user identities (uid, gid, etc), process identities (pid, pgrp, sid, etc), rlimit etc. For check-pointing, we need mainly two components address space and register set.

Since it is often not possible to checkpoint everything that can affect the program behavior, it is essential to identify what is included in a checkpoint in order to guarantee a successful recovery. Figure 2.11 shows the three components which together determine the program behavior. Volatile state consists of the program stack and the static and dynamic data Segments. Volatile state also includes those operating system kernel structures that are essential to current program execution, for example, the program counter, stack pointer, open file descriptors, signal masks and handlers. Persistent state includes all the user files that are related to the current program execution. OS environment refers to the resources that the user processes must access through the operating systems, such as swap space, file systems, communication channels, keyboard, monitors, process id assignments, time, etc.

Since the persistent state is often an important part of most long-running applications, we can include all persistent state in the process state, as indicated in Figure 2.11, to guarantee truly consistent check-pointing and transparent recovery. For saving the disk space there is no use to save temporary file as they can be easily constructed; now we can remove some part of persistent state from process state (Figure 2.11). Actually persistent state position is dependent on implementation.

Files and file descriptors introduce several challenges for checkpoint/restart. These two abstractions are fundamental to the correct execution of most applications. All input/output paths are built on files and file descriptors. Since file operations occur frequently, performance is an important concern. Files may be modified between a checkpoint and the corresponding restart (possibly by the application itself if it is restarted twice from the same checkpoint or from a periodic checkpoint after a fault). Worse still, there are no straightforward ways to know what an application has done to the file system in the interval between checkpoints. If a file descriptor has been

closed, or a file has been unlinked, there are no data structures available to recover the state of the file! Partial solutions to this problem exist, but they involve saving hidden copies of all files when they are opened, and also when the process is check-pointed. Even here, correct execution is impossible to guarantee. There are a few techniques that may be practical. File descriptors are the only link between a process and the corresponding file. File descriptors associated with normal files should be reattached to those files when the application is restarted. File descriptors associated with a terminal (for standard IO) should be attached to the terminal where the restart is requested. Any flags set for the file descriptor, the access mode, and the file offset must be restored after restart. Library implementation moves the system call redirection to maintain a separate copy of the kernel's file descriptor table. In some cases, such as read-only access, the file descriptor table may simply be restored during restart.

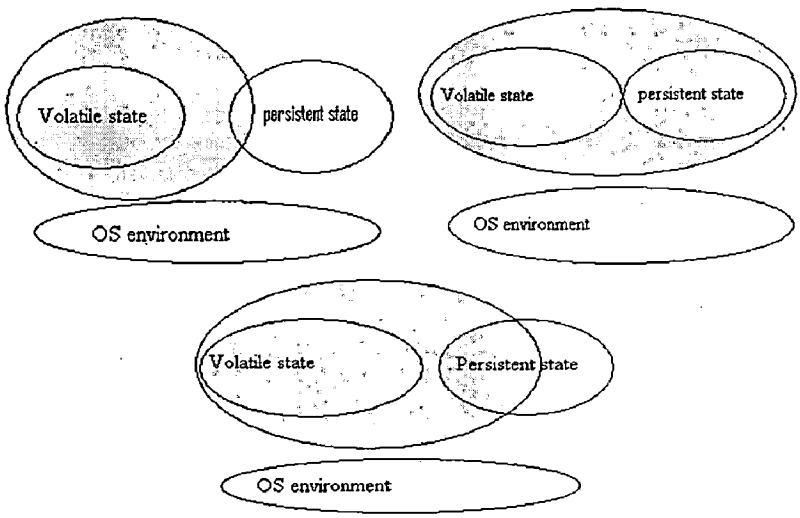


Figure 2.11 Process state (shaded area) vs. process environment (non-shaded area)

- **Fault tolerance with low level check-pointing packages**

Check-pointing gives very good result in fault tolerance. In the case of any failure, check-pointed application can be recovered from its last check-point. There are many low-level [11] and high level check-pointing packages available in market. Each check-pointing package offers different type of services and interfaces. Because of some technical issues check-pointing packages are application dependent. For example in case of distributed application a significant problem is how to take check-

point of a few co-operating processes and not lose the just in-transit message simultaneously. Check-pointing packages have some application issues, which have to consider. These approaches are under consideration of researches due to nice performance in grid environment. A low-level check-pointing package is presented below:

Components

A check-pointing application has following components:

Low-level Check-pointer

Functionality of this component is to take system-level check points. Many low-level check-pointers are available in market but AltrixC/R is famous one due to its unique features. It is a kernel-level check-pointing package design by PSNC for Altrix systems. The most recent version works with Linux kernel 2.6. This package comes with dynamic loader so it is easy to install. This package is able to check-point multi-process programs that communicate through IPC objects.

Execution Manager

The main task of execution manager is to provide its facilities to GRB with uniform interface to various computing nodes. It provides interface to different type of clusters and even with single system. WS GRAM is one of the Executing Manager used now a day. WS GRAM stands for Web Service Grid Resources Allocation and Management and is a part of Globus Toolkit.

Local Resource Manager

The component that provides access to local computing resources is named as Local Resource Manager (LMR). Torque is a Local Resource Manager which is an open source. LRM basically provides control over the jobs distributed among computing nodes of a cluster.

Grid Resource Broker

The Grid Resource Broker is a component that coordinates resource allocation and especially job submissions in Grid environment. This is the component to which end user interacts directly, in order to submit, monitor and control their jobs. This

interface may be GUI, CUI, WWW or WAP-based page. Grid Resource Management Services of GRIDGE Grid Toolkit is famous one due to its unique feature of ability to deal with jobs defined as set of tasks with precedence relationship, where the execution of a child task can be triggered by any status of a parent task.

User Interface

The interface is key component of any application. The interface should provide a neat interacting platform. Apart from its main functionality of controlling and mentoring the jobs, the interface should provide the possibility of presenting the intermediate computing results.

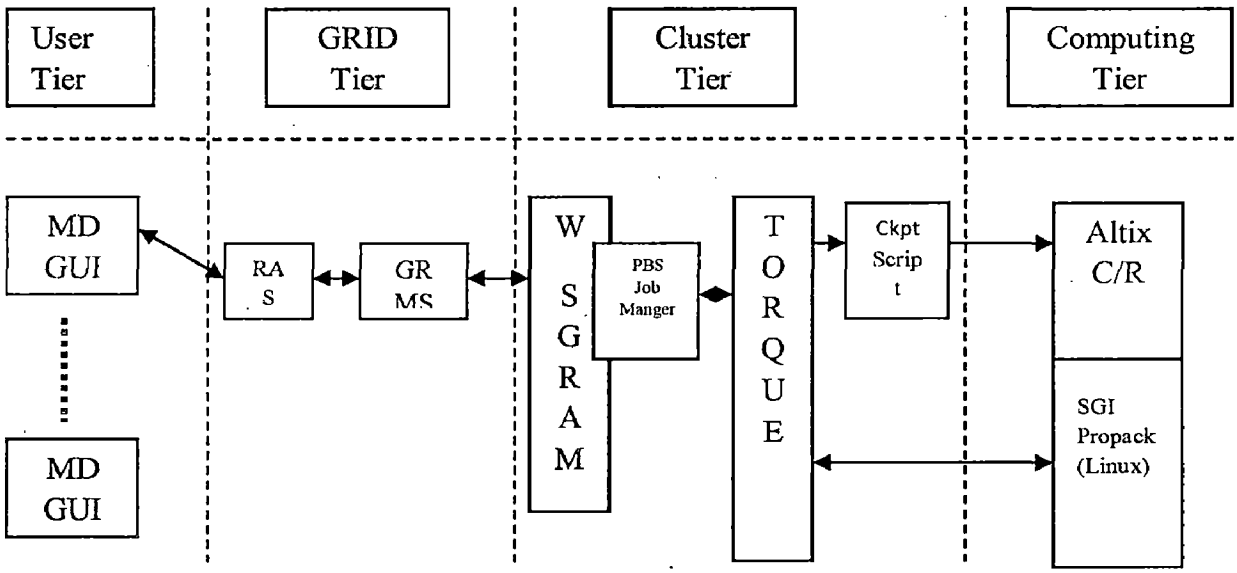


Figure 2.12: Architecture of low level check-pointing packages

- **Adaptive Check-pointing Strategy for Economy based Grid**

The economy based grid is a user centric, resource management and job scheduling approach. It offers incentive and profits to resource owners as award of contributing their resources. On the other hand, it also provides user a flexible environment to maximize their goal within their budget by relaxing QoS and deadline. In this way, economy based grid provide benefit to both of the parties i.e. resource provider and resource consumer. Fault tolerance in such environment is critical to consider because it effect the profit of both the parties and in grid, it become more important because the possibility of faults in grid environment is much higher than a traditional distributed system due to lack of centralized environment, predominant execution of

long jobs, highly dynamic resource availability, diverse geographical distribution of resources, and heterogeneous nature of grid resources.

Grid jobs are executed by economy based grid as follows:

1. Grid user submits their job to Grid Resource Broker (GRB) by specifying their QoS requirements like deadline and budget.
2. GRB schedules user job on best available resource that can fulfill the requirement of user, and job is executed there.
3. The result of job is submitted to user upon successful completion of its job.

Such an economy based environment has following drawbacks:

1. If a fault occurs at grid resource then job is rescheduled and failed to satisfy the QoS requirements like budget and deadline because resubmitted job takes more time and more budget.
2. In such environment there are resources that fulfill the criteria of QoS but have more tendencies toward fault. But GRB again and again selects these resources which makes the scenario worst.

So an adaptive check-pointing [12] fault tolerance approach is used in this scenario to overcome these drawbacks.

System Model

The effect of fault in grid on economy based environment is explained in above description. Faults mainly affect the resource management strategy. The main aim of this model is to optimize the user centric metrics (like number of task executed within and with exceed deadline and budget) in the presence of fault. Here fault occur means a grid resource is unable to complete the task within given time and budget. When such fault is detected by the Grid Resource Broker (GRB), the fault occurrence information of that resource is updated. This fault occurrence information is used during decision making of allocating the resources to the job. This is implemented as fault index. Fault index is maintained and updated when an allocated job completes. This fault index indicates the resource vulnerability to fault i.e. higher the fault index is, higher the failure rate. The fault index of a resource increases every time when the

resource fails to complete the assigned task within deadline and budget. Similarly the fault index decreases every time when resource successfully completes the assigned job within deadline and budget.

This model has the same components as in low level check-pointing packages, but with some additional one. The components are Grid Resource, Fault Tolerance Schedule Manager, Grid Resource Broker, Grid Information Service (GIS) etc. When GRB receive a grid job from the user, it gets the contact information of available grid recourse from the GIS and then contacts with resources and tells them to send their current workload condition. Based on current workload condition of the resources, it prepares a list of resources that can execute the task with user required QoS. Then GRB collects the fault index of selected resources from the Fault Tolerance schedule manager. Depending on the fault index of the resources GRB implements the following Algorithm to take the appropriate decision.

Algorithm for Scheduling of job:

• F: Fault index of the selected grid resource

• $F(i)$, i = 0, 1, 2, . . . ,N, are integers such that $F(0) < F(1) < \cdots < F(N)$

• $C(i)$, i= 1, 2, . . . ,N, are the percentage of task completed such that

$0 <= C(i) <= 100$ and $C(1) > C(2) > \cdots > C(N)$

Step 1: if F(i) <= F < F(i+1) then queue the job and take checkpoint after C(i)

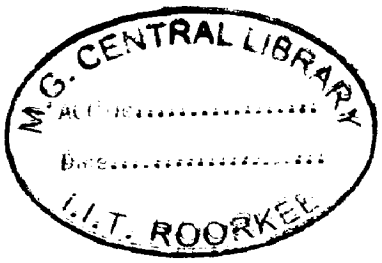
Step 2: (A) if F(N) <= F Then remove the resource from the list and mark it as unavailable.

(B) Get the last check point and re submit the job

Step 3: Update the fault index.

The fault index of grid resource is updated by Fault Tolerance Scheduling Manager using following algorithm.

Algorithm for Updating Fault Index:



Step 1: If Task is completed then

F >= 1 then decrement the fault index.

Step 2: If Task Failure Occur then

i) Increase the fault index.

ii) Resubmit the job with last check-point.

Chapter 3

Resubmission Based Approach

3.1 Working Principle

A process is said to be successfully executed on a resource if and only if it starts when the resource is up and finishes before resource goes down. If a process finish time is greater than resource failure start time, then the process is called as failed. The Grid jobs (Gridlet) are large in size, so they require more computing time, which increases the probability of failure of job. As the execution time of a process is large, larger is the probability that its finish time is greater than resource failure start time.

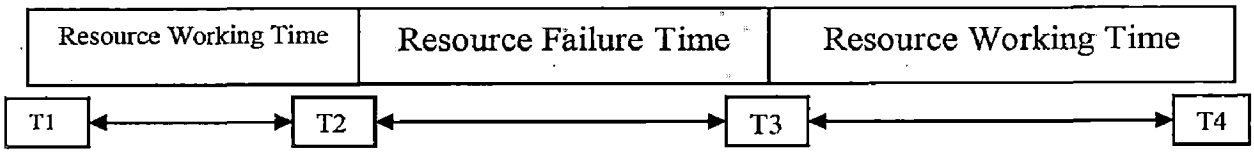


Figure 3.1: Resource life time

Table 3.1 Relationship between Job Status and Resource Life Time

Job Status	Job Start Time(St)	Job End Time(Et)
Success	T1 < St <T2	Et<T2
Failure	T1<St	Et>T2
Success	T3<St<T4	Et<T4

The Resubmission Based approach divides a large task into number of small size subtasks, which can be executed on grid resources and execution of all the subtasks have similar effect as execution of large task. Some of the subtasks are independent to each other, can be executed in parallel to decrease turnaround time which helps in achieving deadline QoS factor.

3.2 System Model

System consists of following components:

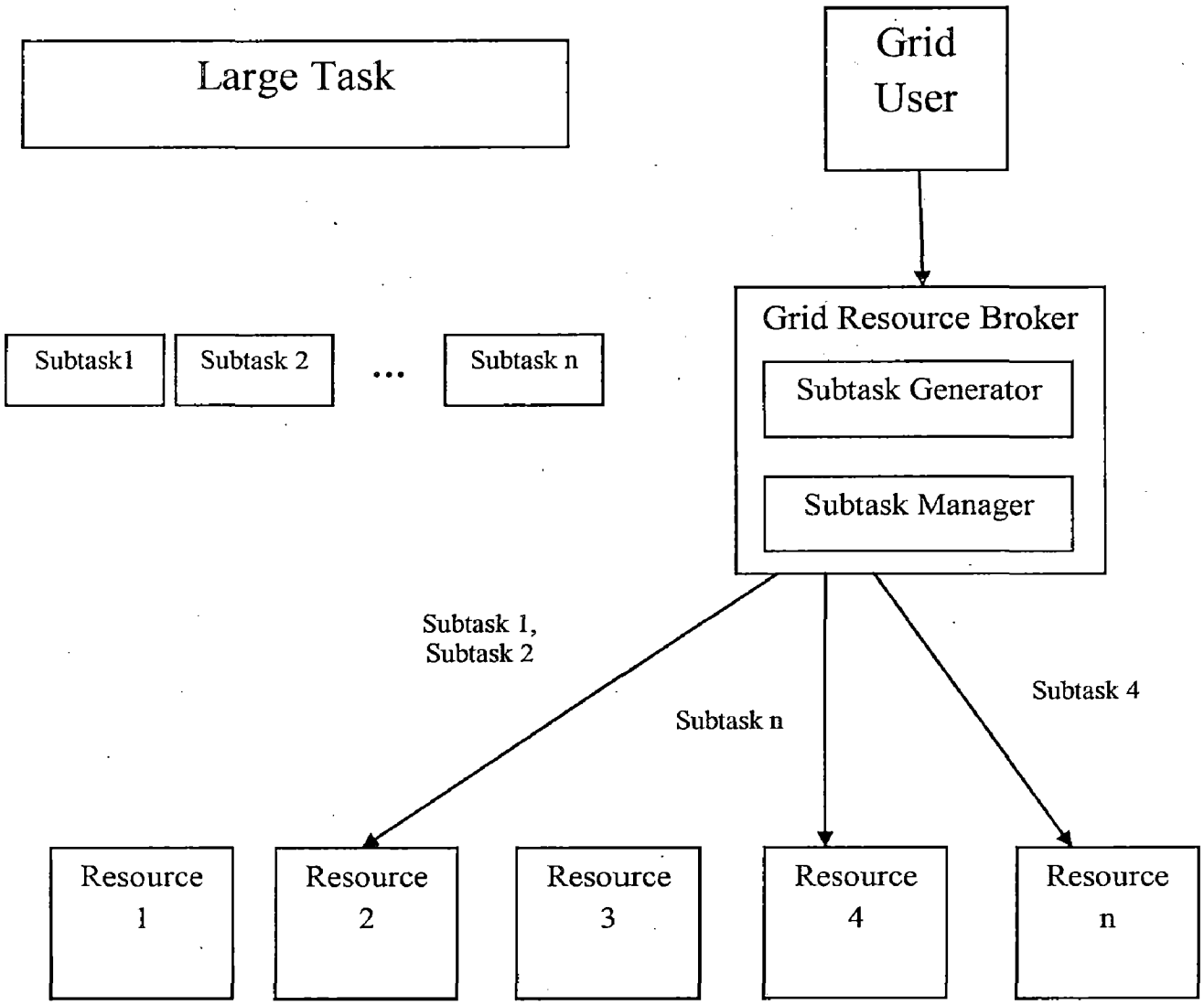


Figure 3.2: System Architecture

- **Grid User:** This component submits Grid jobs (Gridlets) into grid environment. The Grid is constructed to fulfill the requirement of grid user. Grid user is the key component for which grid is constructed. Grid looks like a super computer to the grid user. Grid user thinks, he is the only user which is using the grid, so that each user is not aware of existence of other user. Grid user directly communicates to Grid Resource Broker for executing its large size jobs. Grid Broker returns the results of jobs after execution of job.

- **Grid Resources Broker:** This is the main component, which is responsible for providing a virtual image of super computer to grid user by hiding internal details. It receives Gridlets from grid user and executes them on grid infrastructure and returns back the result of Gridlets. Grid Resource Broker first communicates to Grid Information Service to get the information of grid resources in the grid. Grid Information Service contains all static information of all resources which may be local or global to it after getting the information about all resources. Grid Resource Broker selects one of them on the basis of scheduling algorithm used, and assigns the grid job to that resource and uses heart beat detection approach to detect failure of resource. Grid resource returns the result of job after execution completed and Grid Resource Broker hands over the results to Grid user.

- **Subtask Generator:** It is a part of Grid Resource Broker who deals with dividing the large size task into small size subtasks and constructs the flow graph for subtasks. It also gives a guarantee that execution of all subtasks result in same effect as execution of large size task. Subtask Generator first divides the task into basic block on the basis of control flow dependency. Then it starts picking up each basic block and divides them into subtasks on the basis of data flow dependency. Subtask size checker is a one part of subtask generator. Subtask size is an important factor, which must be considered in this approach. If the subtask size is too large then it does not fulfil the working principle and if the subtask size is too small then most of time is spent in scheduling the subtask, which increases the overhead as well as execution time of the task. Subtask upper size limit and lower size limit are two metrics which are used to decide the subtask size. If subtask size is less then subtask lower limit then small subtasks at same level of flow graph are merged to make a subtask whose size lies between the limits. If the subtask size is larger than the subtask upper limit then subtask is divided into smaller and equal size sub-subtasks so that each sub-subtask's size lies between the limits. All this functionality is done inside subtask size checker.

- **Subtask Manager:** It is also a part of Grid Resource Broker which manages execution of subtasks. It decides which subtask has to execute on which

resource and at what time. It also maintains the sequence of execution of subtasks in such a way that execution of all subtasks result in same effect as execution of large size task. In short this part works as the scheduler of subtasks. It schedules the subtasks on the basis of flow graph. It is also responsible for generation of output after all subtasks successfully executed. If a particular resource fails, it resubmits all the currently executing subtasks on that resource to some other available resource.

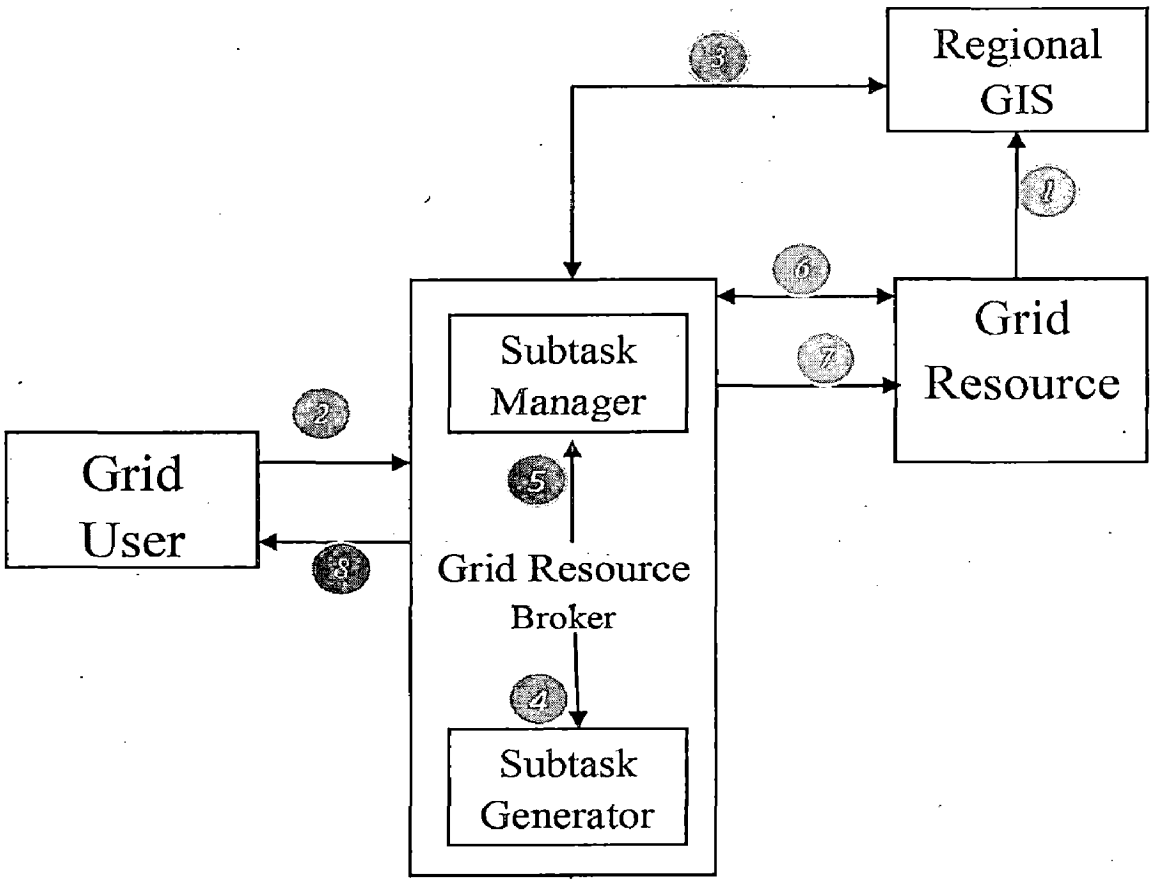


Figure 3.3: Interaction of different components

- **Grid Resource:** This is the component, on which tasks are executed. A grid resource consists of multiple processing elements. Grid resources are geo-graphically distributed. Each grid resource is connected with a Grid Information Service. When a new grid resource is connected or a failed resource is recovered, first of all it communicates with Grid Resource Information Service to register itself. It provides its static information like number of processing elements, speed of processing elements, cost of resource etc. Static information of a resource is such information regarding a resource which does not change over a short time of period. Grid Resource Broker

directly communicates to grid resource and assigns grid jobs on them and takes back the result which Grid Resource Broker hands over to grid user.

Figure 3.3 shows interaction between different components of the model. It tells the order in which different components interact with each other when a grid job is submitted in grid. The explanation of interaction of different components is as follows:

1. Each resource registers itself to the designated GIS to inform its availability and its current static information.
2. Grid user sends job, deadline time and initial budget to the Grid Resource Broker.
3. Grid Resource Broker queries GIS for the available list of resources.
4. Then Grid Resource Broker calls its Subtask Generator to divide the large task into small subtask depending on the control and data flow dependency.
5. Then Grid Resource Broker calls its Subtask Manager to schedule the subtasks.
6. Subtasks are submitted to grid resource for execution.
7. Resource executes the subtasks and results are sent back to the Grid Resource Broker.
8. Grid Resource Broker hand-over the results to Grid User.

3.3 Algorithms

3.3.1 Subtask Generator

Step 1: Large tasks are divided into basic blocks depending on the control flow dependency.

Step2: Repeat step3 to step6 for each instruction in every basic block.

Step3: Compare the input variables with input set and output variables with output set of each subtask generated for basic block in the system till now.

Step4: If no match is found than the instruction does not depend on any existing subtask, so a new subtask is created with that instruction and input set is initialized

with input variables of the instruction and output set is initialized with output variables of the instruction.

Step5: If only one match is found then it means the instruction depends upon the instruction of matched subtask, so that instruction is appended in matched subtask and input variables and output variables are added in input set and output set of the matched subtask.

Step6: If more than one match is found then it means instruction depends upon more than one subtask and a common child node is found in flow graph and instruction is added in that node.

Step7: Flow graph and input and output set of subtasks are returned.

3.3.2 Subtask Manager

Step 1: Repeat the step 2 to step 9 until all subtasks are executed.

Step 2: Find the subtasks in the flow graph that does not depend on any of other subtasks.

Step 3: Repeat step 4 to step 6 until all independent subtasks are scheduled.

Step 4: Find the resources which can satisfy budget and deadline factor for the subtask.

Step 5: If no such resource is found then set status of the job as cancel and return.

Step 6: Among the shortlisted resources, find the lightly loaded resource and schedule the subtask to that particular resource.

Step 7: Wait for the subtask to be completed.

Step 8: If subtask is successfully executed than remove it from the flow graph and go to step 2.

Step 9: If subtask fails due to resource failure then reschedule the subtask to any other available resource.

Step 10: Check the total execution time of job, if it is less than deadline then set its status as successful otherwise set its status as failed.

3.3.3 Grid Resource Broker

Step 1: Grid Job (Gridlet) is submitted by grid user to grid resource broker (GRB).

Step 2: GRB divides the large task into small subtasks.

Step 3: GRB schedules all independent subtasks to different resources and waits for them to complete and then submits rest of the subtasks according to dependency sequence.

Step 4: Failure of a subtask leads to resubmitting the failed subtask on other resource.

Step 5: GRB maintains all the information about the subtasks of a task and when all of the subtasks are executed then GRB returns the result of the task back to the grid user.

The main strength of our approach lies in step2. In this step, first the jobs are submitted to subtask generator. The subtask generator first constructs basic blocks of the given task on the basis of control dependencies. Then it divides each basic block into subtasks according to their data dependency. Finally all the constructed subtasks are passed to subtask size checker. Here we have defined two thresholds for subtask size. If subtask size is small, it will degrade the performance of grid as lot of communication overhead is associated with the execution of subtask. Hence to alleviate this problem, the size checker combines different subtasks on same level of data dependency graph until size becomes within threshold limit. If the size of subtask is greater than upper threshold, it divides that subtask into continuous sub-subtasks. Finally all the subtasks are submitted to the subtask manager (Step 3). Subtask manager schedules the subtasks of each basic block according to their data dependency graph. If any of the subtasks fails, then we have to resubmit only that particular subtask on another available resource. The benefit in Resubmission Based approach is that the subtask size is small, so we have less computational loss and also no need to have check-pointing as tasks are already sub-divided.

3.4 Salient Features

- Failure of one subtask does not affect the other independent subtasks.
- Only failed subtask is resubmitted, not the whole large task.
- No check-pointing is required because the size of subtask is small enough.
- Task turn-around time is reduced because independent subtask can run in parallel, which helps us in achieving the deadline QoS factor.

Chapter 4

Simulation Tool and Parameters

4.1 GridSim: Grid Modeling and Simulation Toolkit

The GridSim[13, 14] toolkit provides a comprehensive facility for simulation of different classes of heterogeneous resources, users, applications, resource brokers, and schedulers. It can be used to simulate application schedulers for single or multiple administrative domains distributed computing systems such as clusters and Grids. Application schedulers in the Grid environment, called resource brokers, perform resource discovery, selection, and aggregation of a diverse set of distributed resources for an individual user. This means that each user has his or her own private resource broker and hence it can be targeted to optimize for the requirements and objectives of its owner. In contrast, schedulers, managing resources such as clusters in a single administrative domain, have complete control over the policy used for allocation of resources. This means that all users need to submit their jobs to the central scheduler, which can be targeted to perform global optimization such as higher system utilization and overall user satisfaction depending on resource allocation policy or optimize for high priority users. GridSim is better for simulating the grid based algorithms because

- It allows modeling of heterogeneous types of resources.
- Resources can be modeled in two modes: space shared and time shared.
- Resource capability can be defined in the form of MIPS (Million Instructions per Second) as per SPEC (Standard Performance Evaluation Corporation) benchmark.
- Advance reservation of resources can be done.
- Application tasks can be heterogeneous and they can be CPU or I/O intensive.
- There is no limit on the number of application jobs that can be submitted to a resource.
- Multiple user entities can submit tasks for execution simultaneously in the same resource, which may be time-shared or space-shared.
- Network speed between resources can be specified.
- It supports simulation of both static and dynamic schedulers.

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

4.2 GridSim Entities

GridSim supports entities for simulation of single processor and multiprocessor, heterogeneous resources that can be configured as time or space shared systems. It allows setting their clock to different time zones to simulate geographic distribution of resources. It supports entities that simulate networks used for communication among resources. During simulation, GridSim creates a number of multi-threaded entities, each of which runs in parallel in its own thread. An entity's behavior needs to be simulated within its body () method, as dictated by SimJava. GridSim based simulations contain entities for the users, brokers, resources, information service, statistics, and network based I/O as shown in Figure 4.1

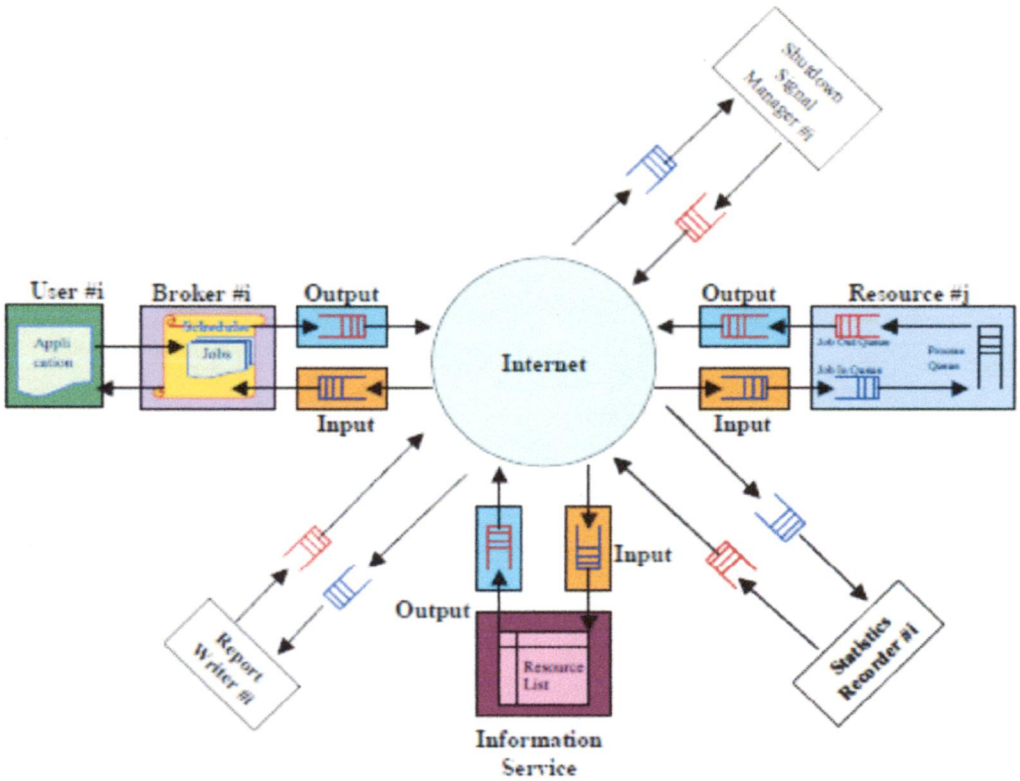


Figure 4.1: A flow diagram in GridSim based Simulation

i) **User:** - Each instance of the User entity represents a Grid user. Each user may differ from the rest of the users with respect to the following characteristics:

- Types of job created e.g., job execution time, number of parametric replications, etc.
- Scheduling optimization strategy e.g., minimization of cost, time, or both,
- Activity rate e.g., how often it creates new job,
- Time zone, and
- Absolute deadline and budget, or
- D-and B-factors, deadline and budget relaxation parameters, measured in the range [0, 1] express deadline and budget affordability of the user relative to the application processing requirements and available resources.

ii) **Broker:** -Each user is connected to an instance of the Broker entity. Every job of a user is first submitted to its broker and the broker then schedules the parametric tasks according to the user's scheduling policy. Before scheduling the tasks, the broker dynamically gets a list of available resources from the global directory entity. Every broker tries to optimize the policy of its user and therefore, brokers are expected to face extreme competition while gaining access to resources. The scheduling algorithms used by the brokers must be highly adaptable to the market's supply and demand situation.

iii) **Resource:** - Each instance of the Resource entity represents a Grid resource. Each resource may differ from the rest of resources with respect to the following characteristics:

- Number of processors;
- Cost of processing;
- Speed of processing;
- Internal process scheduling policy e.g., time shared or space shared;
- Local load factor; and
- Time zone.

The resource speed and the job execution time can be defined in terms of the ratings of standard benchmarks such as MIPS and SPEC. They can also be defined with respect to the standard machine. Upon obtaining the resource contact details from the Grid information service, brokers can query resources directly for their static and dynamic properties.

iv) **Grid Information Service:** - It provides resource registration services and maintains a list of resources available in the Grid. This service can be used by brokers to discover resource contact, configuration, and status information.

v) **Input and Output:** - The flow of information among the GridSim entities happens via their Input and Output entities. Every networked GridSim entity has I/O channels, which are used for establishing a link between the entity and its own Input and Output entities. Note that the GridSim entity and its Input and Output entities are threaded entities i.e., they have their own execution thread with body() method that handle the events. The use of separate entities for input and output enables a networked entity to model full duplex and multi-user parallel communications. The support for buffered input and output channels associated with every GridSim entity provides a simple mechanism for an entity to communicate with other entities and at the same time enables the modeling of a communication delay transparently.

4.3 Application Model

GridSim does not explicitly define any specific application model. It is up to the developers (of schedulers and resource brokers) to define them. In GridSim, each independent task may require varying processing time and input files size. Such tasks can be 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 expressed in MI (million instructions), disk I/O operations, the size of input and output files, and the job originator. These basic parameters help in determining execution time, the time required to transport input and output files between users and remote resources, and returning the processed Gridlets back to the Originator along with the results. The GridSim toolkit supports a wide range of Gridlet management protocols and services that allow schedulers to map a Gridlet to a resource and manage it throughout the life cycle.

4.4 Resource Model

In the GridSim toolkit, we can create Processing Elements (PEs) with different speeds (measured in either MIPS or SPEC-like ratings). Then, one or more PEs can be put together to create a machine. Similarly, one or more machines can be put together to create a Grid resource. Thus, the resulting Grid resource can be a single processor, shared memory multiprocessors (SMP), or a distributed memory cluster of computers. These Grid resources can simulate time- or space-shared scheduling depending on the allocation policy. A single PE or SMP type Grid resource is typically managed by

time-shared operating systems that use round-robin scheduling policy for multitasking. The distributed memory multiprocessing systems (such as clusters) are managed by queuing systems, called space-shared schedulers, that execute a Gridlet by running it on a dedicated PE when allocated. The space-shared systems use resource allocation policies such as first-come-first-served (FCFS), back filling, shortest-job-first served (SJFS), and so on. It should also be noted that resource allocation within high-end SMPs could also be performed using the space-shared schedulers.

4.5 Simulation Environment and Data

Table 4.1 Simulation Parameters

Resource Name	*(Location)*	Nodes	Rating	Policy	GIS	Cost (₹)	Mean Time to Failure
RAL	*(UK)*	41	4900	Space-Shared	2	490	60
Imp.	*College*	52	6200	Space-Shared	2	620	100
NorduGrid	*(Norway)*	17	2000	Space-Shared	2	200	340
NIKHEF	*(Netherlands)*	18	2100	Space-Shared	0	210	340
Lyon	*(France)*	12	1400	Space-Shared	0	140	450
CERN	*(Switzerland)*	59	7000	Space-Shared	0	700	75
Milano	*(Italy)*	5	7000	Space-Shared	1	700	55
Torino	*(Italy)*	2	300	Time-Shared	1	30	130
Rome	*(Italy)*	5	600	Space-Shared	1	60	110
Padova	*(Italy)*	1	100	Time-Shared	1	10	150
Bologna	*(Italy)*	67	8000	Space-Shared	1	80	

- The Grid consists of eleven resources. Their capabilities, mean time to failure and cost are given in Table 4.1.
- Cost of the resources are in cost per million of instructions.

- During simulation 20 Users submit 10 jobs per user.
- Job size is uniformly distributed in [700,000 to 800,000] Million of instruction (MI) units. Its input and out file size is also uniformly distributed in [300 to 500] kilo bytes.
- Resource failure is exponential distributed with mean time of failure of each resource is given in table II. Recovery of failed resource is also simulated using exponential distribution with mean time of 3 minutes.
- In the experiment there are three virtual organizations and each having a GIS.

Chapter 5

Results and Discussions

5.1 Experiment Results

Subtask generator component receives a large task as input and returns a set of subtasks. It also returns precedence constraints directed acyclic flow graph, input and output set of subtasks. Precedence constraints directed acyclic flow graph is created by considering control dependency and data dependency of each block. Subtask generator first converts large task into three address instruction format and then these three address instructions are provided to dependency calculating unit for further processing. The output provided by this unit is shown below:

```
==========Subtask 0 ==============================

Adjacency List is:  3 4
Parents List is:
Input Set is:  h m
Output Set is:  c
==========Subtask 1 ==============================

Adjacency List is:  4
Parents List is:
Input Set is:  b f
Output Set is:  g
==========Subtask 2 ==============================

Adjacency List is:  3
Parents List is:
Input Set is:  h i
Output Set is:  d
==========Subtask 3 ==============================

Adjacency List is:  6 8
Parents List is:  0 2
Input Set is:  a d f k l
Output Set is:  h m
==========Subtask 4 ==============================

Adjacency List is:  6 8
Parents List is:  0 1
Input Set is:  a c f n
Output Set is:  b c
```

```
==========Subtask 5 ==============================

Adjacency List is:  8
Parents List is:
Input Set is:  f i
Output Set is:  j
==========Subtask 6 ==============================

Adjacency List is:
Parents List is:  3 4
Input Set is:  b i n o
Output Set is:  a h
==========Subtask 7 ==============================

Adjacency List is:
Parents List is:
Input Set is:  k n
Output Set is:  e
==========Subtask 8 ==============================

Adjacency List is:
Parents List is: 3 4 5
Input Set is:  b c d f j l n o
Output Set is:  d f g
```

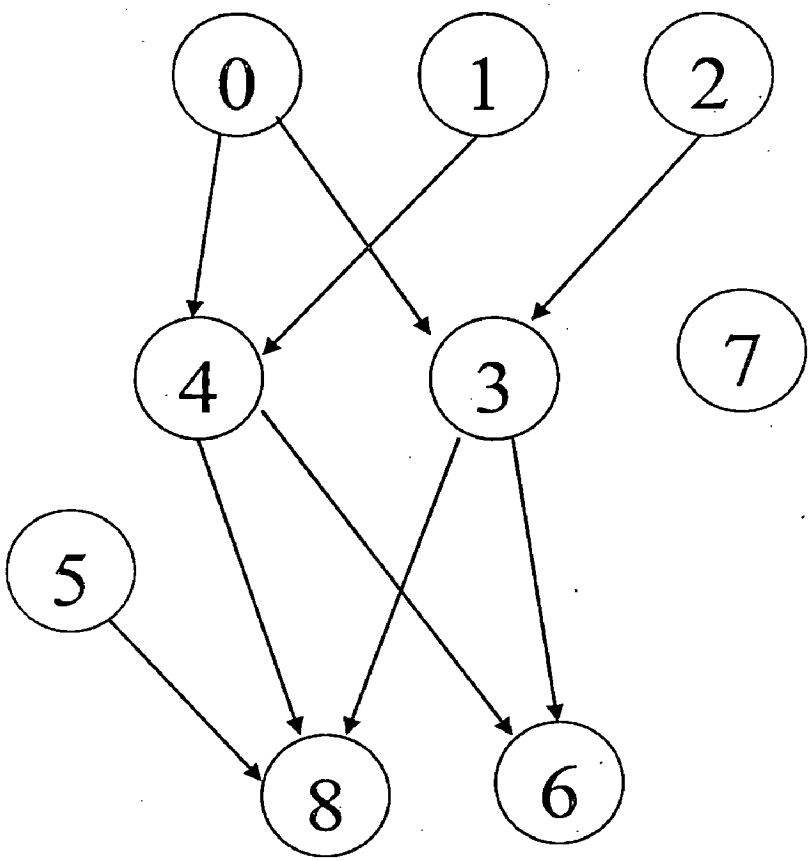


Figure 5.1: Precedence constraints Directed Acyclic Flow Graph.

The output of dependency calculating unit contains Adjacency List, Parents List, Input Set and Output Set of subtasks. Initially the subtasks which are having no parents in Parents List are independent and are scheduled in parallel. After successfully executing a subtask, it is removed from Parents List of the other subtasks which are dependent on this subtask and then a subtask is searched in system which is having empty Parents List.

Figure 5.1 shows the flow graph created by subtask generator unit. The node represents the subtask and a directed edge from i^{th} node to j^{th} node represents that j^{th} subtask is dependent on i^{th} subtask and j^{th} subtask can only be scheduled after the completion of i^{th} subtask.

5.2 Comparison of Check-Pointing approach and Resubmission Based approach

The system models of Check-Point and Resubmission Based approach are designed and tested in GridSim Toolkit-4.0. The GridSim libraries are added to Eclipse. Eclipse is an integrated development environment (IDE) for Java. The GridSim libraries are available freely as java runtime environment (JRE), and they are linked to eclipse platform as an external JRE. A number of resources with different characteristics like cost, CPU rating are used to design grid infrastructure for simulation purpose as mentioned in World Wide Grid (WWG testbed). Different numbers of Gridlets are created to evaluate these approaches. Gridlet is defined in terms of number of instruction (in Million), input file size (in kilo byte), and output file size (in kilo bytes). In the experiment, 200 Gridlets are submitted for different values of budget and deadline for measuring the performance. Gridlets are assigned to two different grids, one in which Check-Point fault tolerance approach is used and to another in which Resubmission Based fault tolerance approach is used. In both the scenarios, first aim is to fulfil the budget and deadline QoS parameter. In Check-Point approach, we take Check-Point at regular intervals.

5.2.1 Different values of budget

Figure 5.2 shows the comparison on the basis of successfully executed gridlets in resubmission based and check-point approach for different values of budget varying

from 5000 to 17000₹, and deadline time is fixed to 120 seconds. Check-Point approach takes Check-Points at regular intervals, which is a time consuming task so some of the tasks do not achieve deadline and are considered as unsuccessful. But in resubmission based approach no Check-Pointing is done and independent subtasks can run in parallel, which helps in achieving deadline hence more jobs are successfully executed in resubmission based approach as compared to Check-Point approach. It is shown in figure 5.2. At the initial stage of the experiment near about 30 jobs get the resources. Rest of 170 jobs fail to satisfy QoS parameters like deadline and budgets, so these 170 jobs are consider as cancelled jobs. Out of 30 jobs in check-point approach near about 20 jobs successfully executed and rest of the 10 jobs failed to complete within deadline time. In resubmission based approach independent subtasks can execute in parallel so all 30 jobs are successfully executed. It also reduces average execution time in resubmission based approach.

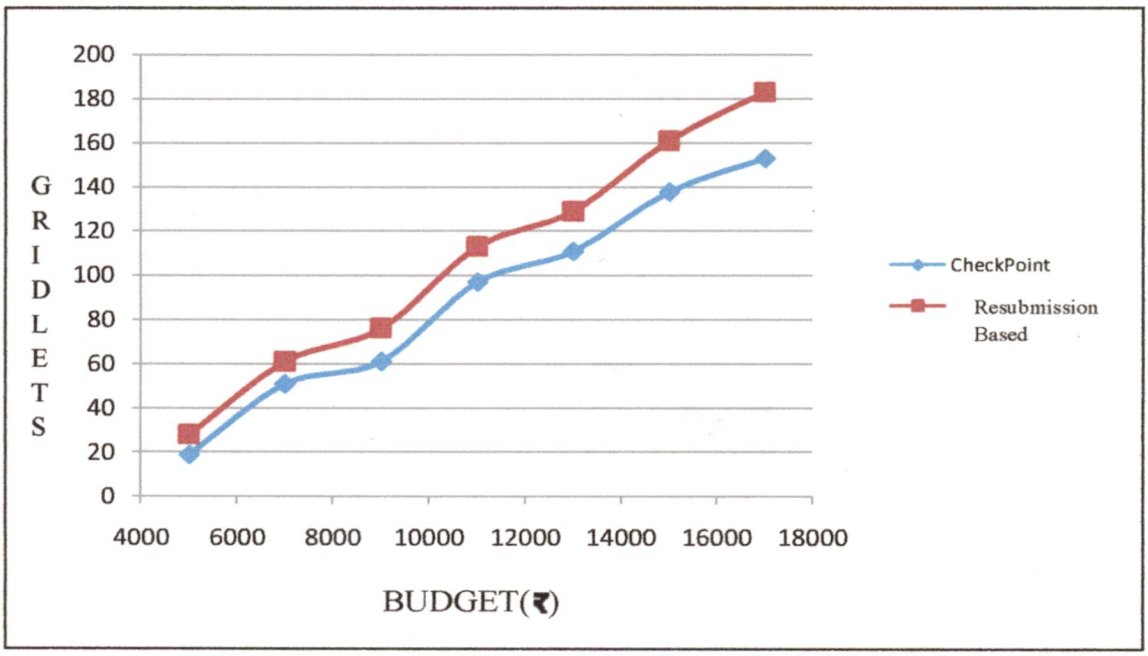


Figure 5.2: Number of job success for different values of budget

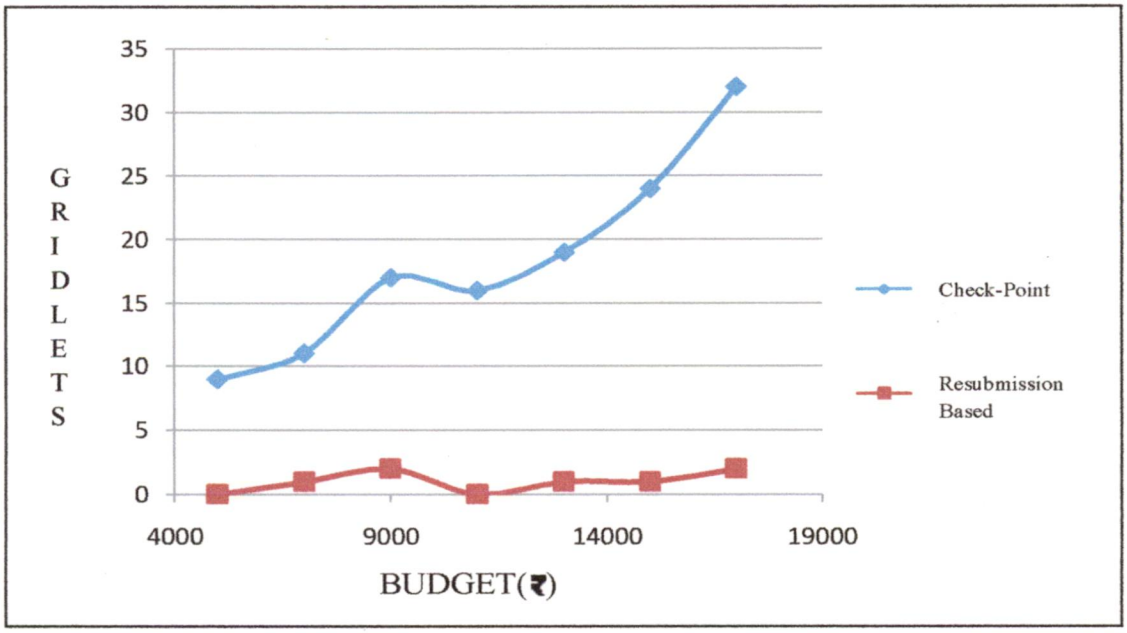


Figure 5.3: Number of jobs fail to achieve deadline

Figure 5.3 shows the number of jobs failed due to deadline time in both approaches. Number of jobs failed to finish within deadline is plotted and it is observed that in resubmission based approach very few jobs failed to finish within deadline time as compared to check-point approach. The check-point consumes time in saving status at each check-point at regular interval which is an overhead in execution of job. Let assume that 1 second is consumed in saving status at each check-point and average 10 check-points are taken during execution of a job then overall 10 second (1 * 10) of extra time is consumed in check-point approach. Other reason for better performance of resubmission based approach is that it divides large task into small subtasks which can be execute in parallel. Due to this parallelism between different subtasks, execution time of a job is lesser as compare to execution time in check-point approach which helps the jobs to finish with in dead line time.

5.2.2 Different values of deadline time

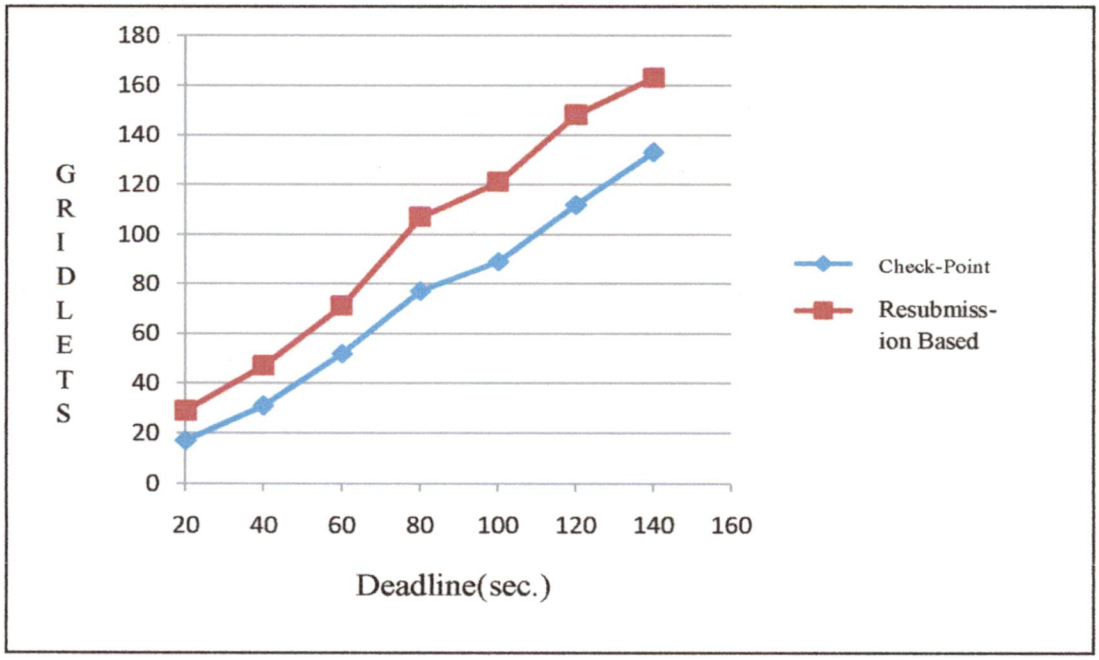


Figure 5.4: Number of job success for different values of deadline

Figure 5.4 shows the comparison in number of gridlet successfully executed in resubmission based and check-point based approach for different values of deadline time which varies from 20 to 150 second and budget is fixed to 12000₹. Some time is consumed in check-pointing which lead to fail a task to complete within deadline. Graph shows that more jobs are completed within deadline time in resubmission based approach as compared to check-point based approach. At the initial stage of the experiment near about 40 jobs get the resources and rest of 160 jobs are failed to satisfy QoS parameters like deadline and budgets, so these 160 jobs are consider as cancelled job. Out of 40 jobs in check- point approach near about 20 jobs successfully executed and rest of the 20 jobs failed to complete within dead line time but in resubmission based approach 30 jobs are successfully executed due to independent subtasks can execute in parallel. 10 jobs are failed in resubmission based approach to finish within deadline time because deadline time is small in initial stage of the experiment.

Chapter 6

Conclusions and Scope for Future Work

6.1 Conclusions

In this dissertation, a novel fault tolerance approach is proposed that aims to successfully execute grid tasks in the presence of resource failure in economy based grid environment. Large grid task are divided into small subtasks to achieve fault tolerance. Independent subtasks can run in parallel which decrease the average execution time of task, which helps in completing the task within deadline time. Resubmission based approach is simulated in GridSim ToolKit-4.0 and compared with check-pointing approach. The experiment results show that resubmission based approach reduces the execution time by running subtasks in parallel so that lesser number of jobs failed to achieve deadline. Resubmission approach is simple and there is no wastage of time for taking Check-Points, which decreases the execution time of a job. Experimental results also show that it provides better fault tolerance environment to grid because more number of jobs are successfully executed as compared to Check-Point approach.

6.2 Scope for Future Work

In the future, this work can be extended in following ways:

i) Better resource allocation policy can be used, which can consider dynamic nature of resources.

ii) This work can be implemented in actual middle ware like Globus.

iii) More independency can be achieved within the subtasks which lead to less execution time.

REFERENCES

[1] M. Baker, R. Buyya and D. Laforenza, "Grids and Grid Technologies for Wide-area Distributed Computing", *Published in Journal of Software-Practice & Experience*, Vol. 32, No.15, pp:1437-1466, Dec 2002.

[2] I. Foster, C. Kesselman, and S. Tuecke. "The anatomy of Grid: Enabling scalable virtual organizations". *Published in the International Journal of Supercomputer Application,* 15(3), Aug,2001.

[3] Klaus Krauter, Rajkumar Buyya, and Muthucumaru Maheswaran, "A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing", *Published in Software: Practice and Experience (SPE) Journal,* ISSN: 0038-0644, vol. 32, Issue 2, 2002, Wiley Press, USA, Feb 2002.

[4] R. Medeiros, W. Cirne, F. Brasileiro, and J. Sauvé, "Faults in Grids: Why are they so bad and What can be done about it?," *In Proc. 4th International Workshop on Grid Computing* , pp. 18-18, Jan 2003.

[5] A. Litke, K. Tserpes, K. Dolkas, and T. Varvarigou, "A task replication and fair resource management scheme for fault tolerant grids," *In Proc. Of Advances in Grid Computing-EGC 2005,* pp. 1022-1031,Jan 2005.

[6] K. Erciyes, "A replication-based fault tolerance protocol using group communication for the grid," *Parallel and Distributed Processing and Applications, Lecture Notes in Computer Science,* vol. 43, pp. 672-681, Aug 2006.

[7] I. Suzuki and T. Kasami, "A distributed mutual exclusion algorithm," *Published in ACM Transactions on Computer Systems (TOCS),* vol. 3, pp. 344-349, Aug 1985.

[8] C. Jiang, C. Wang, X. Liu, and Y. Zhao, "A Fuzzy Logic Approach for Secure and Fault Tolerant Grid Job Scheduling," *Autonomic and Trusted Computing, Lecture Notes in Computer Science,* vol. 46, pp. 549-558, Jan 2007.

[9] Partha Sarathi Mandal, Krishnendu Mukhopadhyaya, "Performance analysis of different checkpointing and recovery schemes using stochastic model", *Published in Journal of Parallel and Distributed Computing*, vol. 66, pp 99-107,2006

[10] P. Riteau, A. Lebre, and C. Morin, "Handling Persistent States in Process Checkpoint/Restart Mechanisms for HPC Systems," *In Proc.* Of *9th IEEE/ACM International Symposium on Cluster Computing and Grid CCGRID '09.*, pp. 404-411, May 2009.

[11] G. Jankowski, R. Januszewski, R. Mikolajczak, P. Supercomputing, N. Center, and J. Kovacs, "Improving the fault tolerance level within the GRID computing environment-integration with the low-level checkpointing packages," *CoreGRID Technical Report Number* TR-0158, June 16 2008.

[12] B. Nazir, K. Qureshi, and P. Manuel, "Adaptive checkpointing strategy to tolerate faults in economy based grid," *Published in the Journal of Supercomputing*, vol. 50, pp. 1-18, May 2009.

[13] GridSim toolkit. [Online] Available: http://www.gridbus.org/gridsim/

[14] Rajkumar Buyya and Manzur Murshed, "GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing, Concurrency and Computation" *Published in Practice and Experience (CCPE)*, vol. 14, Issue 13-15, pp: 1175-1220, ISSN: 1532-0626, Wiley Press, New York, USA, Dec 2002.

LIST OF PUBLICATIONS

[1] Vinit Kumar, Dr. Padam Kumar, "Module Based Approach for Fault tolerance in Grid Environment", *Second International Conference on Advances in Computer Engineering, ACE '11,* 25-26 Aug2011, Trivendram, Kerala, India. [Accepted]