Implementing Checkpointing Algorithm in Alchemi.NET

Thesis submitted in partial fulfillment of the
requirements for the award of
degree of

Master of Engineering
in
Computer Science and Engineering

By:
Neeraj Kumar Rathore
(80632016)

Under the supervision of:
Ms. Inderveer Chana
Senior Lecturer, CSED

Thapar University, Patiala

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

JUNE 2008

Certificate

I hereby certify that the work which is being presented in the thesis titled, **"Implementing Checkpointing Algorithm in Alchemi.NET"**, in partial fulfillment of the requirements for the award of degree of Master of Engineering in Computer Science and Engineering submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of *Ms. Inderveer Chana* and refers other researcher's works which are duly listed in the reference section.

The matter presented in this thesis has not been submitted for the award of any other degree of this or any other university.

(***Neeraj Kumar Rathore)***

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

(**Ms. Inderveer Chana)**
Sr. Lecturer
Computer Science and Engineering Department
Thapar University
Patiala

Countersigned by

(Dr. SEEMA BAWA)
Professor & Head
Computer Science & Engineering Department
Thapar University
Patiala

(**Dr. R.K.SHARMA)**
Dean (Academic Affaris)
Thapar University,
Patiala.

Acknowledgement

I wish to express my deep gratitude to Ms. Inderveer Chana, Senior Lecturer and Dr. (Ms.) Seema Bawa Professor & Head, Computer Science & Engineering Department for providing their uncanny guidance and support throughout the preparation of the thesis report.

I am also heartily thankful to Dr. Maninder Singh, Assistant Professor, Computer Science and Engineering Department and Ms. Damandeep Kaur, P.G. Coordinator, Computer Science and Engineering Department for the motivation and inspiration that triggered me for my thesis work.

I would also like to thank all the staff members, Thapar Grid Group and all my friends especially Vineet, Rohit, Kunal, Lokesh, Kabir, Satyarth ,Gurpreet, Anchal, Rajeev ,Ms Anju and Ms. Shashi who were always there at the need of the hour and provided all the help and support, which I required for the completion of the thesis.

Last but not the least, I would like to thank God for not letting me down at the time of crisis and showing me the silver lining in the dark clouds.

(Neeraj Kumar Rathore)

80632016

Abstract

The Grid is rapidly emerging as the means for coordinated resource sharing and problem solving in multi-institutional virtual organizations while providing dependable, consistent, pervasive access to global resources. The emergence of computational Grids and the potential for seamless aggregation and interactions between distributed services and resources, has led to the start of new era of computing. Tremendously large number and the heterogeneous nature of grid computing resource make the resource management a significantly challenging job. Resource management scenarios often include resource discovery, resource monitoring, resource inventories, resource provisioning, fault isolation, variety of autonomic capabilities and service level management activities.

Out of this fault tolerance has become the main topic of research as till date there is no single system that can be called as the complete system that will handle all the faults in grids.

Checkpointing is one of the fault-tolerant techniques to restore faults and to restart job fast. The algorithms for checkpointing on distributed systems have been under study for years. These algorithms can be classified into three classes: coordinated, uncoordinated and communication-induced algorithms. In this thesis, a checkpointing algorithm that has minimum checkpointing counts equivalent to periodic checkpointing algorithm has been proposed. Relatively short rollback distance at faulty situations and produces better performance than other algorithms in terms of task completion time in both fault-free and faulty situations. This algorithm has been implemented in Alchemi.NET because it did not currently support any fault tolerance mechanism.

Table of Contents

List of Figures

List of Tables

INTRODUCTION 1

Grid is a type of distributed system that supports the sharing and coordinated use of geographically distributed and multi- owner resources independently from their physical type and location in dynamic virtual organizations that share the same goal of solving large-scale applications.

This chapter introduces grid computing, discusses about the history, types, and the motivation of this research work and finally gives the framework of this thesis.

1.1 Grid Computing

Rajkumar Buyya defined the Grid as:

"Grid is type of parallel and distributed system that enables the sharing, selection and aggregation of geographically distributed resources dynamically at run time depending on their availability, capability, performance, cost, user quality-of–self-service requirement"[45]

"*Grid Computing enables virtual organizations to share geographically distributed resources as they pursue common goals, assuming the absence of central location, central control, omniscience, and an existing trust relationship*" [5].

In other words:

- Grid is a service for sharing computer power and data storage capacity over the Internet and Intranet.
- Grid is beyond simple communication between computers but it aims ultimately to turn the global network of computer into one vast computational resource.
- Grid is to coordinate resources those are not subject to centralized control.
- Grid is to use standard, open, general-purpose protocols and interfaces.
- Grid is to deliver nontrivial Qualities of Service.

Grid computing resources has enhanced the performance of computers and reduced their costs. This availability of low cost powerful computers coupled with the popularity of the Internet and high-speed networks has led the computing environment to be mapped from distributed to Grid environments [2]. Moreover, In recent times, rapid advances in networking, hardware and middleware technologies are facilitating the development and deployment of complex applications, such as large-scale distributed, collaborative scientific simulation, analysis of experiments in elementary

particle physics, distributed mission training etc. These predominantly collaborative applications are characterized by their very high demand for computing, storage and network bandwidth requirements [1], which can be fulfilled by Grid Computing. Recent researches on computing architectures have allowed the emergence of a new computing paradigm known as Grid computing.

1.1.1 History of Grid

The term "Grid" was coined in the mid 1990s to denote a proposed distributed computing infrastructure for advanced science and engineering. The concept of Computational Grid has been inspired by the 'electric power Grid', in which a user could obtain electric power from any power station present on the electric Grid irrespective of its location, in an easy and reliable manner. Whenever additional electricity is required, just plug into a power Grid to access additional electricity on demand, similarly for computational resources plug into a Computational Grid to access additional computing power on demand using most economical resources [2].

Table-1.1 Historical Background of the Grid [2]

Technology	**Year**
Networked Operating Systems	1979-81
Distributed operating systems	1988-91
Heterogeneous computing	1993-94
Parallel and distributed computing	1995-96
The Grid	1998

The theory behind "Grid Computing." is not to buy more resources but to borrow the power of the computational resources you need from where it's not being used". Grid has been around in one form or other throughout the history of computing. There is a certain amount of "reinventing the wheel" going on in developing the Grid. However, each time the wheel is reinvented , it is reinvented in a much more powerful form, because computer processors, memories and networks improve at an exponential rate [37]. Because of the huge improvements of the underlying hardware (typically more than a factor of 100x every decade), it is fair to say that reinvented wheels are qualitatively different solutions, not just

small improvements on their predecessor [45].

1.1.2 Types of Grid

Grid computing can be used in a variety of ways to address various kinds of application requirements. Often, the type of solutions categorizes Grids that they best address. Of course, there are no hard boundaries between these Grid types and often Grids may be a combination of two or more of these [11-13].

Grids can be classified on the basis of two factors:

- Scale
- Functionality

On basis of **scale** they can be further classified as:

Cluster Grid: Cluster Grid is simplest form of Grid and provides a compute service to the group or department level.

Enterprise Grid: Enterprise Grids enable multiple project or department to share recourses with in enterprise or campus and not necessary have to address security and other global policy management issues associated with global Grid [15].

Global Grid: Global Grids are collection of enterprise and cluster Grid as well as other geographically distributed resources, all of which are agreed upon global usage policies and protocols to enable resources sharing [14].

On basis of **Functionality** they can be further classified as:

Compute Grids: A compute Grid is essentially a collection of distributed computing resources, within or across locations that are aggregated to act as a unified processing resource or virtual supercomputer.

Data Grids: A data Grid provides wide area, secure access to current data. Data Grids enable users and applications to manage and efficiently use database information from distributed locations.

1.1.3 Benefits of Grid computing

Following are the benefits that Grid computing provides to user community, developer community and enterprise community as well .It provides an abstraction for resource sharing and collaboration across multiple administrative domains [43].

A) Exploit unused resources

In most organizations, computing resources are underutilized. Most desktop machines are busy less than 25% of the time (if we consider that a normal employee works 7 hours a day and that 42 hours a week and that there are 168 hours per week) and even the server machines can often be fairly idle. Grid computing provides a framework for exploiting these underutilized resources and thus has the possibility of substantially increasing the efficiency of resource usages. The easiest use of Grid computing would be to run an existing application on several machines. The machine on which the application is normally run might be unusually busy; the execution of the task would be delayed. Grid Computing should enable the job in question to be run on an idle machine elsewhere on the network [1].

B) Increase Computation

Grids helps o increase the computation power in the following ways [42]:

(a) Hardware Improvement

Microprocessor architecture and other resource capabilities of personal computers continuously increase to provide users with additional power.

(b) Periodic Computational Needs

Some applications only need computational power once in a while. The systems are fully utilized at that times and idle the rest of the time.

(c) Capacity of Idle Machines

Machines are often idle and thus their computational power is free to use. The idea would be to use it only during that idle time and leave once the computer is in use.

(d) Sharing of Computational Results

The key to more sharing may be the development of collaboratories centers without walls, in which the nation's researchers can perform their research without regard to geographical location-interacting with colleagues, accessing instrumentation, sharing data and computational resources, and accessing information in digital libraries. Considering these areas of interest, communication networks in place could act as a medium to provide access to advanced computational capabilities, regardless of the location of resources.

1.2 Motivations and Research Question

Grid Computing enables aggregation and sharing of geographically distributed computational, data and other resources as single, unified resource for solving large-scale compute and data intensive computing application. Management of these resources is an important infrastructure in the grid computing environment. It becomes complex as the resources are geographically distributed, heterogeneous in nature, owned by different individual or organizations with their own policies, have different access and cost models, and have dynamically varying loads and availability. The conventional resource management schemes are based on relatively static model that have centralized controller that manages jobs and resources accordingly. These management strategies might work well in those scheduling regimes where resources and tasks are relatively static and dedicated. However, this fails to work efficiently in many heterogeneous and dynamic system domains like grid where jobs need to be executed by computing resources, and the requirement of these resources is difficult to predict.

Due to highly heterogeneous and complex computing environments, the chances of faults increases, therefore it is necessary to design a mechanism to check for faults and handle them efficiently in such infrastructure. Handling faults (configuration, middleware, application and hardware faults) in highly dynamic networks where the availability of resources changes at a high rate requires a mechanism that should be scalable, adaptable, robust and reliable [3].

The complex, heterogeneous and dynamic systems present new challenges in resource management such as: scalability, adaptability, reliability and fault tolerance. This thesis work focuses on he fault tolerance aspect of grid computing.

1.3 Thesis Framework

This section discusses the framework of this thesis. This thesis is organized as follows:

Chapter 2 reviews the background of grid computing and discusses about virtual organization, Issues, architecture and characteristics and analyzes need for fault tolerance in heterogeneous and dynamic environment such as Alchemi.NET middleware Grids. First part defines the fault tolerant mechanism, types of fault in grid, their characteristics and main design features. Second, it summarizes the kind of checkpoints that exist in grids and various techniques how to deal with kind of faults. Also a survey describing checkpoint algorithm and comparison at certain parameter in different perspective to make middleware

fault tolerance is defined.

Chapter 3 gives the project description. In this we first introduce the Alchemi.NET based grid, why Alchemi.NET middleware was chosen and then give reasons for choosing Alchemi.NET as the framework for this thesis and description about the fault tolerance in different middleware. The second part of this chapter discusses the Fault Tolerance in Alchemi.NET. What fault tolerance mechanisms exist and what still lack in Alchemi.NET for fault tolerance have been discussed. In the last section discusses the challenges and the solution of grid, which came in between the implementation.

Chapter 4 discusses about the proposed technique, implementation requirements, implementation detail, checkpointing algorithm, applications and proposing framework of Alchemi.NET with algorithm and flow chart of Alchemi.NET Executor and Manager.

Chapter 5 discusses the implementation of the new fault tolerance system for the checkpointing based Alchemi.NET grids. This purposed system would try to deal with all the deficiencies present in Alchemi.NET. It also shows the experiment result snap shots of interfaces, databases and tables.

Chapter 6 presents the conclusion of the thesis, describes the main contribution of the thesis and highlights future research direction based on the results obtained.

LITURATURE SURVEY 2

The last decade has seen a considerable increase in commodity computer and network performance, mainly as a result of faster hardware and more sophisticated software. The early efforts in grid computing started as a project to link supercomputing sites, but now it has grown far beyond its original intent [3]. In fact, there are many applications that can benefit from the grid infrastructure, including collaborative engineering, data exploration, high throughput computing, and of course distributed supercomputing.

This chapter gives an introduction to grid computing. The first section, discusses about the grid architecture, concept of virtual organization and faults in grid. Next fault tolerance and checkpointing mechanisms of grid has been explored.

2.1 Grid Computing

The rapid and impressive growth of the internet, there has been a rising interest in web-based parallel computing. In fact, many projects have been incepted to exploit the Web as an infrastructure for running coarse-grained distributed parallel applications. In this context, the web has the capability to become a suitable and potentially infinite scalable metacomputer for parallel and collaborative work as well as a key technology to create a pervasive and ubiquitous grid infrastructure[5].

This section will discuss the grid architecture and the details of Virtual Organization.

2.1.1 Grid architecture

The Architecture principles for a Global Grid have to be established on the following principles, which summarized below:

- Employ a common network infrastructure
- Transport any traffic type
- Integration of various transport media
- Adaptation to changes
- Provide Quality of Service

The architecture of the Grid is often described in terms of “layers”, each providing a specific function [17]. In general, the higher layer is user-centric, whereas the lower layers are more hardware-centric (Figure 2.1).

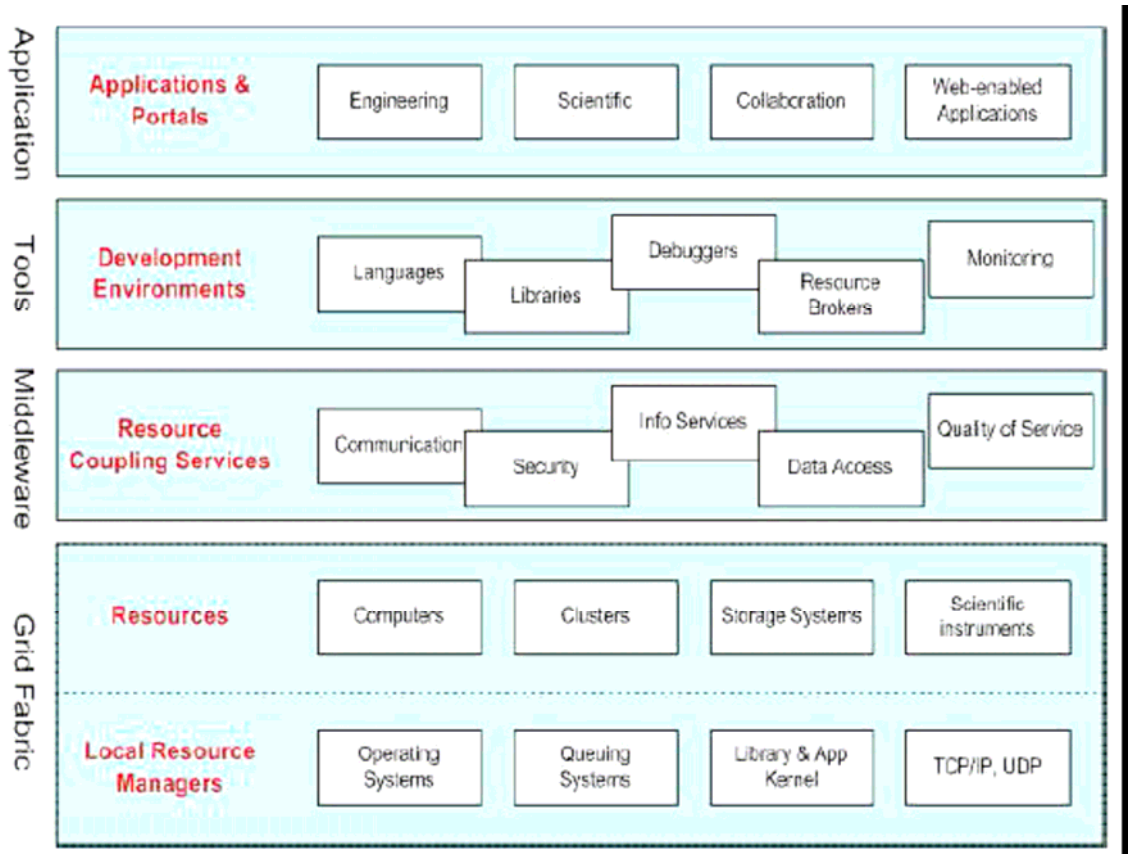


Figure 2.1 Layers of Grid Architecture [17]

A) Network Layer

At the bottom is the Network Layer, which assures the connectivity for the resources in the Grid. On top of it lies the Resource Layer, made up of the actual resources that are part of the Grid, such as computers, storage systems, electronic data catalogues, and even sensors such as telescopes or other instruments, which can be connected directly to the network.

B) Middleware Layer

The Middleware Layer provides the tools that enable the various elements (servers, storage, networks, etc.) to participate in a unified Grid environment. The Middleware layer can be thought of as the intelligence that brings the various elements together.

C) Tool Layer

The Tool Layer is made up of tools that assure connectivity between Middleware and applications. It deals with several languages using a set of libraries to integrate the requests

either from lower or higher layers.

D) Application Layer

The highest layer of the structure is the Application Layer, which includes all different user applications (science, engineering, and business, financial), portals and development toolkits supporting the applications. In this layer of the Grid, grid user will "see" and most of the time interacts through their browser. This layered structure can be defined in other ways. For example, the term fabric is often used for all the physical infrastructure of the Grid, including computers and the communication network. Within the Middleware layer, distinctions can be made between a layer of resource and connectivity protocols, and a higher layer of collective services. However, in all schemes, the Applications Layer remains the topmost layer.

2.1.2 Concept OF Virtual Organizations (VO)

The concept of a Virtual Organization is the key to Grid Computing [1]. It is defined as a dynamic set of individuals and/or institutions defined around a set of resource- sharing rules and conditions.All VOs share some commonality among them, including common concerns and requirements, but may vary in size, scope, duration, sociology, and structure [8].

The members of any VO negotiate on resource sharing based on the rules and conditions defined in order to share the resources from the thereby automatically constructed resource pool. Assigning users, resources, and organizations from different domains across multiple, worldwide geographic territories to a virtual organization are one of the fundamental technical challenges in Grid Computing. This complexity includes the definitions of the resource discovery mechanism, resource sharing methods, rules and conditions by which this can be achieved, security federation and/or delegation, and access controls among the participants of the VO. This challenge is both complex and complicated across several dimensions.

It undeniably creates the illusion of a simple large and powerful self-managing virtual computer, which gives the opportunity to use and reach heterogeneous systems sharing massive resources as shown in figure 2.2.

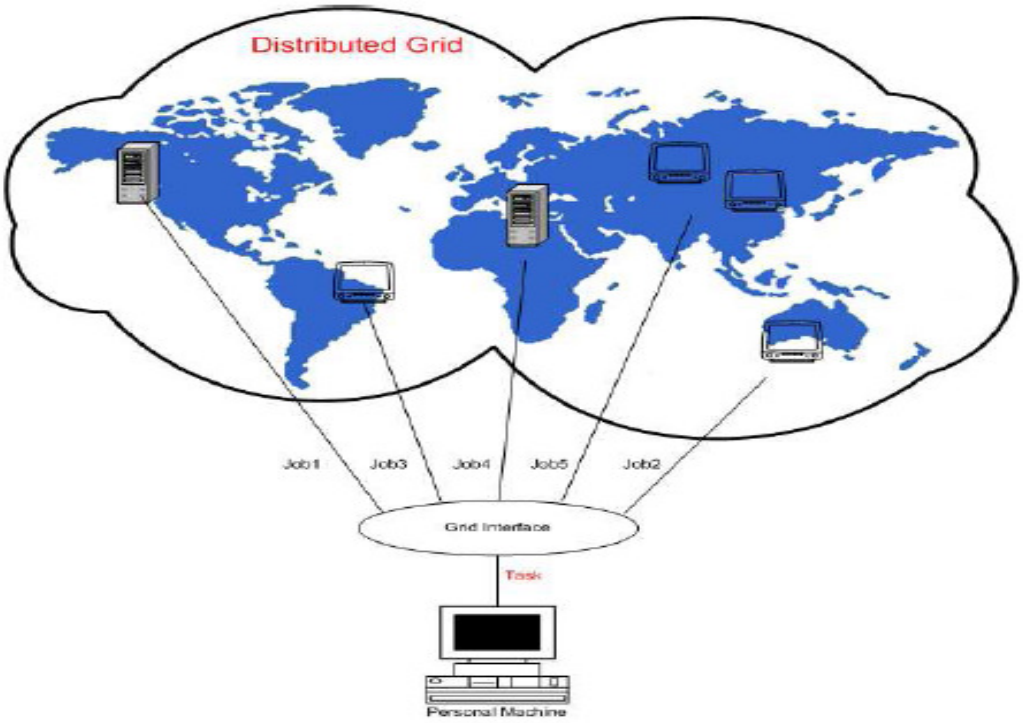


Figure 2.2: A Grid members submits a task to the Grid via the Grid Interface [8]

Virtual organizations can span from small corporate departments that are in the same physical location to large groups of people from different organizations that are spread out across the globe. Virtual organizations can be large or small, static or dynamic [7].

2.1.3 FAULTS IN GRID

As VO involve in a large number of complex nature of heterogeneous networks, In which faults is a major concern. When a fault occurs choose various ways for detection of such occurrences can be accomplished. For this it is important to:

- Rapidly determine exactly where the fault is,
- Isolate the rest of the network from the failure so that it can continue to function without interference,
- Reconfigure or modify the network in such a way as to minimize the impact of operation without the failed component or components, and
- Repair or replace the failed components to restore the network to its

initial state.

As grid is a distributed and unreliable system involving heterogeneous resources located in different geographical domain, for this case fault tolerant resource allocation services [25] have to be provided. In particular, when crashes occur, tasks have to be reallocated quickly and automatically, in a completely transparent way from the user's point of view.

Thus, one of the main challenges for grid computing is the ability to tolerate failure and recover from them (ideally in a transparent way). Current grid middleware still lacks mature fault tolerant features and next generation grids needs to solve this problem providing a more dependable infrastructure to execute large-scale computations by using remote clusters and high performance computing system. A resource management system for the grid should address the issues of fault tolerance.

Fault Avoidance

Traditionally, software dependability has been achieved by fault avoidance techniques (such as structured programming and software reuse) in order to prevent faults, or fault removal techniques (such as testing) to detect and delete faults. However, in the case of Grid computing, these approach – although still very much useful but may not be enough. As applications scale to take advantage of Grid resources, their size and complexity will increase dramatically. However, experience has shown that systems with complex asynchronous and interacting activities are very prone to errors and failures due to their extreme complexity, and simply cannot expect such applications to be fault free no matter how much effort is invested in fault avoidance and fault removal. In fact, the likelihood of errors occurring may be exacerbated by the fact that many Grid applications will perform long tasks that may require several days of computation, if not more. In addition to this, it may also be the case that the cost and difficulty of containing and recovering from faults in Grid applications is higher than that of normal applications. Furthermore, the heterogeneous nature of Grid nodes means that many Grid applications will be functioning in environments where interaction faults are more likely to occur between disparate Grid nodes, whilst the dynamic nature of the Grid – resources may enter and leave at any time, in many cases outside of the applications control – means that a Grid

application must be able to tolerate (and indeed, expect) resource availability to be fluid [53].

2.2 FAULT TOLERANT TECHNIQUES

In a grid environment there are potentially thousands of resources, services and applications that need to interact in order to make possible the use of the grid as an execution platform. Since these elements are extremely heterogeneous, there are many failure possibilities, including not only independent failures of each element, but also those resulting from interactions between them. Because of the inherent instability of grid environments, fault-detection and recovery is a critical component that must be addressed. The need for fault-tolerance is especially acute for large parallel applications since the failure rate grows with the number of processors and the duration of the computation[14].

The earliest use of computers made it apparent that even with careful design and good components, physical defects and design errors were unavoidable. Thus, designers of early computers used practical techniques to increase reliability. They used redundant structures to mask failed components; error control codes and duplication or triplication with voting to detect or correct information errors; diagnostic techniques to locate failed components and automatic switchovers to replace failed subsystems.

2.2.1 Fault Tolerance Definitions

Fault tolerance is the survival attribute of computer systems. The function of fault tolerance is

"…to preserve the delivery of expected services despite the presence of fault-caused errors within the system itself. Errors are detected and corrected, and permanent faults are located and removed while the system continues to deliver acceptable service [20]".

Fault tolerance is the property of a system that continues operating consistent with its specifications even in the event of failure of some of its parts [21]. From a user's point of view, a distributed application should continue despite failures. The fault tolerance has become the main topic of research. Till now there is no single system that can be called as the complete system that will handle all the faults in grids. Grid is a dynamic system and the nodes can join and leave voluntarily. For making fault tolerance system a

success, following point must consider:

- How new nodes join the system,
- How computing resources are shared,
- How the resources are managed and distributed

2.2.2 Fault Tolerance in Grid Middleware

Grids have middleware stacks, which are a series of cooperating programs, protocols and agents designed to help users access the resources of a Grid [36]. Grid Middleware refers to the security, resource management, data access, instrumentation, policy, accounting, and other services required for applications, users, and resource providers to operate effectively in a Grid environment. Middleware acts as a sort of 'glue', which binds these services together. Middleware connect applications with resources.

Formally Grid middleware can be defining [23] as:

"A mediator layer that provide a consistent and homogeneous access to resources managed locally with different syntax and access methods"

Fault-tolerance or graceful degradation is the property that enables a system (often computer-based) to continue operating properly in the event of the failure of (or one or more faults within) some of its components. If its operating quality decreases at all, the decrease is proportional to the severity of the failure, as compared to a naively-designed system in which even a small failure can cause total breakdown. Fault-tolerance is particularly sought-after in high-availability or life -critical systems [40].

Recovery from errors in fault-tolerant systems can be characterized as either roll-forward or roll-back. When the system detects that it has made an error, roll-forward recovery takes the system state at that time and corrects it, to be able to move forward. Roll-back recovery reverts the system state back to some earlier, correct version, for example using checkpoint and moves forward from there. Roll-back recovery requires that the operations between the checkpoint and the detected erroneous state. [40].

The brief overview of the some popular middleware Alchemi.NET, Globus, Sun Engine and Condor and their fault tolerance mechanisms has been discussed in this section.

A) Alchemi.NET

Alchemi.NET [34] is a .NET-based grid computing framework that provides the runtime machinery and programming environment required to construct desktop grids and

develop grid applications. Till today not much work is done on Alchemi.NET Fault tolerance. The basic technique used by Alchemi.NET for Fault Tolerance is Heart beating. The Executors in the Alchemi.NET sends the heartbeat signals to the Manager at regular interval of time. Manager after receiving signals from executors guesses that the executor node is still working. So the procedure of fault tolerance is not so well planned. In later chapters we will see fault tolerance procedure of Alchemi.NET in more detail and try to find challenges in its fault tolerance system and how they can removed using different techniques like checkpointing etc .

B) Condor

Condor is a high-throughput distributed batch computing system. It is a specialized workload management system for compute-intensive jobs. It provides a job queuing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management. Users submit their serial or parallel jobs to Condor, Condor places them into a queue, chooses when and where to run the jobs based upon a policy, carefully monitors their progress, and ultimately informs the user upon completion [35].

Fault tolerance in the condor is the complete set of information that comprises a program's state. Given a checkpoint, a program can use the checkpoint to resume execution. For long running computations, the ability to produce and use checkpoints can save days, or even weeks of accumulated computation time. If a machine crashes, or must be rebooted for an administrative task, a checkpoint preserves computation already completed. Condor makes checkpoints of jobs, doing so periodically, or when the machine on which a job is executing will shortly become unavailable. In this way, the job can be continued on another machine (of the same platform); this is known as process migration [33]. Condor provides high support for authorization, authentication, and encryption. When the Condor is installed the default configuration setting includes none of them. The administrator, through the use of macros, enables these features.

C) Sun Grid Engine

Sun’s Grid Engine is a Distributed Resource Management tool; it provides load management across heterogeneous, distributed computing environments. Sun made the source available under the Grid Engine project [28]. The Grid Engine project is an open

source version of the software is known as Grid Engine. Grid Engine is generally installed across a cluster of machines using a shared file system. It can be configured to work across disparate file systems. Users submit jobs to Grid Engine and Grid Engine manages the allocation of jobs to machines within the cluster. This ensures the resources are used more productively therefore increasing availability**.** In this middleware with the help of the user and kernel level Checkpointing manage the fault [52].

D) Globus

The Globus [32] toolkit is designed to enable people to create computational Grids. Globus is an open source initiative aimed at creating new Grids capable of the scale of computing seen only in supercomputers up to now. As an open source project any person can download the software, examine it, install it and hopefully improve it. By this constant stream of comments and improvements, new versions of the software can be developed with increased functionality and reliability. In this way the Globus project itself will be on going with constant evolution of the toolkit [16]

Globus Toolkit has three pyramids of support built on top of a security infrastructure,

as illustrated. They are:

• Resource management

• Data management

• Information services

All of these pyramids are built on top of the underlying Grid Security Infrastructure (GSI). This provides security functions, including single/mutual authentication, fault tolerant, confidential communication, authorization, and delegation [32].

Through the monitoring of the system by the gate keeper we can manage the fault. It provide secure communication between clients and servers**.** It also communicates with the GRAM client (globusrun) and authenticates the right to submit jobs. After authentication, gatekeeper forks and creates a job Manager delegating the authority to communicate with clients [16].

2.2.3 Fault Tree Analysis

Because of computational Grid heterogeneity, scale and complexity, faults become likely. Therefore, Grid infrastructure must have mechanisms to deal with faults while also

providing efficient and reliable services to its end users [42].

The fault tree analysis classifies faults that may take place in Grid Computing [42]. In the figure 2.3 various kinds of faults that can occur have been shown.

There are mainly six classes of faults as discussed below:

1) Hardware faults:

Hardware failures take place due to faulty hardware components such as CPU, memory, and storage devices [50].

- CPU faults arise due to faulty processors, resulting in incorrect output.
- Memory faults are errors that occur due to faulty memory in the RAM, ROM or cache.
- Storage faults occur for instance in secondary storage devices with bad disk sectors.
- Components that are used beyond specification.

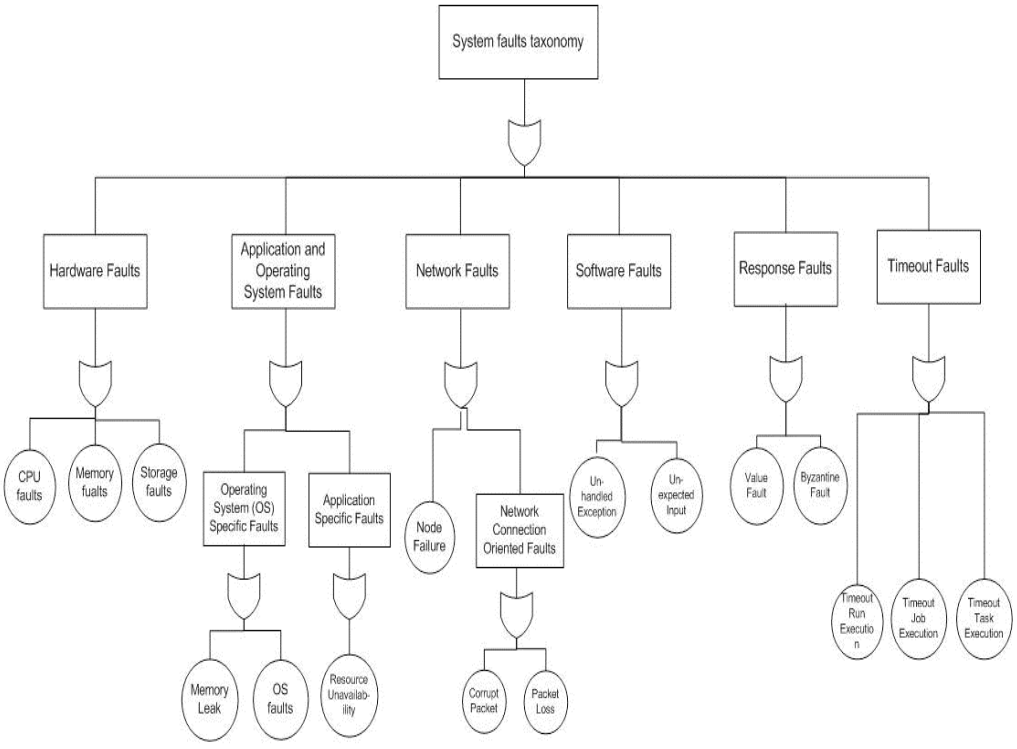


Figure 2.3 Fault Tree Analyses in Grid Computing [42].

2) Application and operating system faults:

Application and operating system failures occur due to application or operating system specific faults.

- Memory leaks are an application specific problem, where the application consumes a large amount of memory and never releases it.
- Operating system faults include deadlock or inefficient or improper resource management.
- Resource unavailable: Sometimes an application fails to execute because of resource unavailability, that is, the need for a resource is use by other applications.

3) Network faults:

In Grid computing, resources are connected over multiple and different types of distributed networks. If physical damage or operational faults in the network will occur then the network may exhibit significant packet loss or packet corruption, as a result individual nodes in the network or the whole network may go down [41].

- Node failure: In node failure, individual nodes may go down due to operating system faults, network faults or physical damage.
- Packet loss: Broken links or congested networks may result in significant packet loss.

Corrupted packet: Packets can be corrupted in transfer from one end to another.

4) Software faults:

In a particular task several high resource intensive applications usually run on Grid and while running this software application several software failures take place. Such as:

- Un-handled exception: Software faults occur because of un-handled exception like divide by zero, incorrect type casting etc.
- Unexpected input: Operators may enter incorrect or unexpected values into a system that causes software faults, for example specifying an incorrect input file or invalid location of an input file

5) Response faults:

Several kinds of response faults like the following can occur:

- Value fault: If some lower level system or application level fault has been overlooked (due to unknown problem) an individual processor or application may emit incorrect output.

- Byzantine error: Byzantine errors take place due to failed or corrupted processors that behave arbitrarily.

6) Timeout faults:

Timeout faults are higher level faults that take place due to some lower level faults at the hardware, operating system and application, software or network.

2.2.4 Fault Tolerance Mechanisms

In addition to ad-hoc mechanisms – based on users complaints and log files analysis – grid users have used automatic ways to deal with failures in their Grid Environment. To achieve the automatic ways to deal with failures, various fault tolerance mechanisms are there. Some of these fault tolerance mechanisms are:

- *Application-dependent:* Grids are increasingly used for applications requiring high levels of performance and reliability, the ability to tolerate failures while effectively exploiting the resources in scalable and transparent manner must be integral part of grid computing resource management systems[24].
- *Monitoring Systems*: In this a fault-monitoring unit is attached with the grid. The base technique which most of the monitoring units follow is heartbeating technique [25]
- *Fault Tolerant Scheduling:* With the momentum gaining for the grid computing systems, the issue of deploying support for integrated scheduling and fault-tolerant approaches becomes a paramount importance [27]. For this most of the fault tolerant scheduling algorithms are using the coupling of scheduling policies with the job replication schemes such that jobs are efficiently and reliably executed. Scheduling policies are further classified on basis of time-sharing and space sharing.
- *Checkpointing-recovery:* Checkpointing and rollback recovery provides an effective technique for tolerating transient resource failures, and for avoiding total loss of results. Checkpointing involves saving enough state information of an executing program on a stable storage so that, if required, the program can be re-executed starting from the state recorded in the

checkpoints. Checkpointing distributed applications is more complicated than Checkpointing the ones, which are not distributed. When an application is distributed, the Checkpointing algorithm not only has to capture the state of all individual processes, but it also has to capture the state of all the communication channels effectively [26].

2.3 Checkpointing: Fault Tolerance Mechanism

Checkpointing is a technique for inserting fault tolerance into computing systems. It basically consists on storing a snapshot of the current application state, and uses it for restarting the execution in case of failure. It is saving the program state, usually to stable storage, so that it may be reconstructed later in time. Checkpointing provides the backbone for rollback recovery (fault-tolerance), playback debugging, process migration and job swapping. It mainly focuses on fault-tolerance, process migration and the performance of checkpointing on all computational platforms from uniprocessors to supercomputers.

Checkpointing and restart has been one of the most widely used techniques for fault tolerance in large parallel applications. By periodically saving application status to permanent storage (disk or tape), the execution can be restarted from the last checkpoint if system faults occur. It is an effective approach to tolerating both hardware and software faults. The best strategy depends on the criteria for optimization and saving of program state, usually to stable storage, so that it may be reconstructed later in time. It also provides the backbone for rollback recovery (fault-tolerance), playback debugging, process migration and job swapping. For example, a user who is writing a long program at a terminal can save the input buffer occasionally to minimize the rewriting caused by failures that affect the buffer. Check pointing and Rollback techniques offer interesting possibilities to achieve fault tolerance without appreciable cost and complexity increment. It has useful feature for both availability and minimizing human programming effort.

2.3.1 Checkpointing Definitions

Local checkpoints*:* A process may take a local checkpoint any time during the execution. The local checkpoints of different processes are not coordinated to form a global consistent checkpoint [30].

Forced checkpoints: To guard against the domino effect [6], a CIC protocol piggybacks protocol specific information to application messages that processes exchange. Each process examines the information and occasionally is forced to take a checkpoint according to the protocol.

Useless checkpoints: A useless checkpoint of a process is one that will never be part of a global consistent state [31]. Useless checkpoints are not desirable because they do not contribute to the recovery of the system from failures, but they consume resources and cause performance overhead Checkpoint intervals: A checkpoint interval is the sequence of events between two consecutive checkpoints in the execution of a process

Recovery line*:* The system is then required to roll back to the latest available consistent set of checkpoints, which is also called the recovery line, to ensure correct recovery with a minimum amount of rollback

Fault Tolerance*:* The ability of a system to perform with a fault present, achieved through reconfigurations, passive fault tolerance, or graceful degradation.

Passive Fault Tolerance: An implementation of fault tolerance such that no action is necessary for the system to perform in the presence of a fault. An example of passive fault tolerance is multiple connections between a module and a backplane of the same signal. One of these connections may fail, and the system will still fully operate.

Reconfiguration*:* Action taken in response to a fault resulting in either a controlled loss of functionality or no loss of functionality.

Graceful Degradation*:* A type of fault tolerance where the system is now operating in a mode with less than full functionality, but with some controlled level of functionality. An example of graceful degradation is the dynamic modification of an algorithm in response to a failure of the sensor.

Redundancy*:* A manner to implement reconfiguration in which some resources are kept as spares or backups, either hot, warm, or cold, so that they may be used in the event of a failure. Examples of redundancy range from simple spare modules to designing the system with more capability than is necessary so that the extra capability may be used in the event of a failure Dual (or Full) Redundancy .

2.3.2 Checkpointing Schemes

This section describes the existing checkpointing schemes:

Application checkpointing

It is dominant approach for most large-scale parallel systems which requires the application programmer to insert checkpoint code to checkpoint library functions at appropriate points in the computation. It is mainly used to insert checkpoint in the computation when the "live" state is small, to minimize the amount of state that has to be stored [18].

System checkpointing

It can save entire computational state of the machine and time, (which is determined by the large-scale system). It requires more saving state than application checkpoints, this means system can checkpoint any application at an arbitrary point in its execution, but allows programmers to be more productive [18].

System-initiated provides transparent, coarse-grained checkpointing, which is better to save the kernel level information and runtime policy whereas application-initiated provides more efficient, portable, fine-grained checkpointing, Which is aware the semantics of data where application state is minimal and used for compiler optimization.

Cooperative Checkpointing

Cooperative checkpointing is a set of semantics and policies that allow the application compiler and system to jointly decide when checkpoints should be performed. Specifically, the application requests checkpoints, which have been optimized for performance by the compiler and the system can grant or deny these requests.

Application and system-initiated checkpointing each have their own pros and cons. Excepts transparency, Cooperative checkpointing confers nearly all the benefits of the two standard schemes. In the absence of better compilers or developer tools, however, transparency necessarily comes at the cost of smaller, more efficient checkpoints that are not an acceptable tradeoff for most high performance applications. This is useful to construct reliable systems [18].

2.3.3 Phases of Checkpointing

Checkpointing has two phases:

- Saving a checkpoint and
- Checkpoint recovery following the failure.

To save a checkpoint, the memory and system, necessary to recover from a failure is sent to storage. Checkpoint recovery involves restoring the system state and memory from the checkpoint and restarting the computation from the checkpoint was stored. When the time lost in computation is called overhead time that has to save a checkpoint and restore a checkpoint after a failure and the re-computation is performed after checkpoint but before the failure. However, this loss contributes to the computer unavailability so is better and beneficial, instead of restarting job after occurrence of a failure. It applies both to system and application checkpoint. Some of the basic action is given below: -

Checkpoint saving steps: -

- First of all, we decide checkpoint on the basis of the application code, scheduler and some policies.
- After that all processors must reach a safe point, which includes appropriate handling of outstanding reads and writes at that time. Detected errors should be handled before initiating a checkpoint.
- Then a copy of where memory for system checkpoints is stored.
- Then, is committed checkpoint it means there where no errors during data copy, so that the checkpoint can be safely used for recovery and
- At last we can resume computing of from the processor safe points.

Checkpoint recovery steps: -

- First of all, we determine need for recovery from a failure and rerun a computation from a certain point where we select most recent checkpoint.
- After that halt the processors and reset them to a know state.
- Then, we update system configuration to determine resources to use for failure recovery, e.g., a spare processor to use in place of a failed processor.
- Next, we copy all memory for system checkpoint from storage and commit checkpoint recovery where need to be certain where no errors during data copy.
- At last, resume computing from the recovered system state.
- Each action may involve multiple messages responses and context switches. The decision to commit a checkpoint or checkpoint recovery requires successful acknowledgment from all the system resources involved [19].

2.3.4 Checkpointing Types

Their ae following types of checkpointing :

A) Disk Based checkpointing

B) Disk Less checkpointing

C) Double checkpointing

A) Disk Based Checkpointing:

In checkpoint based methods, the state of the computation as a checkpoint is periodically saved to stable storage, which is not subject to failures. When a failure occurs the computation is restarted from one of these previously saved states. According to the type of coordination between different processes while taking checkpoints, checkpoint-based methods can be broadly classified into three categories:

i) Uncoordinated checkpointing,

ii) Coordinated checkpointing and

iii) Communication-induced checkpointing.

Uncoordinated Checkpointing

In this checkpointing, each process independently saves its checkpoints for a consistent state from which execution can resume. However, it is susceptible to rollback propagation, the domino effect [6] possibly cause the system to rollback to the beginning of the computation. Rollback propagations also make it necessary for each processor to store multiple checkpoints, potentially leading to a large storage overhead .

Coordinated Checkpointing

It requires processes to coordinate their checkpoints in consistent global state. This minimizes the storage overhead, since only a single global checkpoint needs to be maintained on stable storage. Algorithms used in this approach are blocking [28] (used to take system level checkpoints and non-blocking (uses application level checkpointing) [9].It does not suffer from rollback propagations.

Communication-induced Checkpointing

In this checkpointing the processes works in a distributed environment where it takes independent checkpoint to prevent the domino effect by forcing the processors to take

additional checkpoints based on protocol-related information piggybacked on the application messages from other processors [43].

B) Diskless Checkpointing

It is a technique for distributed system with memory and processor redundancy. It requires two extra processors for storing parity as well as standby. Process migration feature has ability to save a process image. The process can be resumed on the new node without having to kill the entire application and start it over again. It has memory or disk space .In order to restore the process image after a failure, a new processor has to be available to replace the crashed processor. This requires a pool of standby processors for multiple unexpected failures [22].

The comparison between disk based and disk less checkpointing for distributed and parallel system in certain parameter is described in table 2.1.

Table 2.1: On disk and Disk less checkpointing for parallel and distributed system [22]

Parameter	**Disk Based**	**Diskless**
Latency time	High	Low
CPU overhead	High	High
Memory requirement	Low	high
Stable storage requirement	High	Low
Toleration of wholesale failure	Yes	No
Reliability	High	Low
Efficiency	Low	High
Addition hardware	Not Required	Additional processors
Portability	High	Low

C) Double Checkpointing

Double checkpointing targets on relatively small memory footprint on very large number of processors when handles fault at a time, each checkpoint data would be stored to two different locations to ensure the availability of one checkpoint. In case the other is lost using two buddy processors have identical checkpoints. It can be stored either in the memory or local disk of two processors. These are double in-memory checkpointing and double in-disk checkpointing schemes. In this scheme, store checkpoints in a distributed fashion to avoid both the network bottleneck to the central server [58].The comparison

between Disk-based and Memory-based Checkpoint in certain parameter is described in table 2.2.

Table 2.2: Comparison of Disk-based and Memory-based Checkpoint Schemes [39]

Fault tolerant protocols	**Double in Memory**	**Double in Disk**
Shrink/Expand	Yes	Yes
Portability	Low	Low
Foolproof	No	No
Diskless	Yes	No ,local disk
Halts job	No	No
Bottleneck	No	No
Require backup processors	Not Necessarily	Not Necessarily
Transparent checkpoint	No	No
Synchronized checkpoint	Yes	Yes
Automatic restart	Yes	Yes

A) Double In-memory Checkpointing

In this checkpointing each process stores its data to memory of two different processors. It has faster memory accessing capability, low checkpoint overhead and faster restart to achieve better performance than disk-based checkpoint. But it will increase the memory overhead and initiate checkpointing at a time when the memory footprint is small in the application. This can be applied to many scientific and engineering applications such as molecular dynamics simulations that are iterative.

B) Double In-disk Checkpointing

It is useful for applications with very big memory footprint where checkpoints are stored on local scratch disk instead of in processor memory. Due to the duplicate copies of checkpoints it doesn't rely on reliable storage. It incurs higher disk overhead in checkpointing but does not suffer from the dramatic increase in memory usage as in the double in-memory checkpointing. Taking advantage of distributed local disks, it avoids the bottleneck to the central fileserver [39].

2.3.5 Comparison between Different Checkpoint Schemes

Based on the literature survey following comparisons have been made: -

Table -2.3 Comparative study of different checkpointing schemes

Check pointing Methods	Uncoordinated Check pointing	Coordinated Check pointing	Communication Induced	Diskless check pointing	Double Check pointing
Efficiency	High for small process	Low	Low	High	High
Performance	Low	Low	Low	Higher for distributed applications	Faster
Portability	High	High	High	Low	Low
Cost	High	Low, negligible for low memory usage application	High	High	Very High
Scalable	No	Minimal	Not scale for large number of processors	Difficult to scale to large number of processors	Highly
Flexibility	Low	All processes Save their states at the same time	Process can be moved from one node to another by writing the process image directly to a remote node	Replace stable storage with memory and processor redundancy	Handle fault at a time and Availability of one checkpoint in case the other is lost.
Overhead	Large Storage, Very high Log management and Work in small Process	Minimum storage overhead And negligible overheads in failure-free Executions.	High latency and Memory and disk overhead	High memory overhead for storing checkpoints	Low memory overhead
Advantages	Most Convenient and Save their checkpoints Individually	Not suffer from rollback propagations and Processes Save their state together	Preventing domino effect, piggybacking And information Of regular message exchanged by the processes	Improve performance in distributed /parallel applications and Process migration save process image	Uses in small memory footprint on large number of processors. ex-scientific applications
Recovery	Checkpoint Of the faulty process is restored	Processes stop regular message activity to take their checkpoints and coordinated way to analyze and restore the last set of Checkpoints	Needed large number of forced checkpoints nullify the benefit of autonomous local checkpoints Using new process to restore the process image after failure	Using parity/backup and extra processors for storing parity as well as replace failed application processors.	Through automatic restart and synchronization by two identical checkpoint buddy processors to provide foolproof fault tolerance
Disadvantages	Unsuitable, Domino Effect, Wastage memory ,unbounded & complex Garbage collection	Consistent checkpoint and Large latency for saving the checkpoints storage	Deteriorated parallel performance & Requires standby processors	Communication bottleneck	Depend on a central reliable storage and required Additional Hardware

2.4 Summary

This chapter introduced grid computing. Section 1 focused on Architecture, Virtual organization and Faults of grid. Section 2 started with the introduction of fault tolerance definitions, middleware faults, mechanism and the factors that are to be considered while constructing the Alchemi.NET fault tolerance. At last in Section 3 different types of checkpointing technique, phases, types have been discussed. On the comparison basis a better checkpoint technique in different grid environment has been discussed. Research finding and problem formulation in brief has been discussed in next chapter.

RESEARCH FINDINGS 3

As described earlier that Grid environments are featured by an increasingly growing virtualization and distribution of resources. Such situations impose greater demands on fault-tolerance capabilities .In the fault tolerance mechanisms different types of schemes and techniques are there, which are beneficial to make grid fault tolerant. Checkpointing is one of the techniques of fault tolerance. Till now many middleware in the grid environment are not fully fault tolerant. Different middleware have different levels of fault tolerance. Some of the middleware like Alchemi.NET do not have a robust fault tolerance mechanism. Therefore, in this research work Alchemi.NET has been chosen and a checkpointing algorithm has been designed for it.

This chapter is organized as follows: The section 3.1 discusses about the general Alchemi.NET based Grids why we need for this and the selection of Alchemi.NET as framework for establishing the Computational Grid Environment. The section 3.2 of this chapter discusses the Fault tolerance with some present scenarios in Alchemi.NET. At last in Section 3.3 discusses the brief introduction of the problem formulation.

3.1 Why Alchemi.NET?

Alchemi.NET is an open source software framework that allows you to painlessly aggregate the computing power of networked machines into a virtual supercomputer And develop applications to run on the grid. Alchemi.NET includes:

- The runtime machinery (Windows executables) to construct grids.
- A .NET API and tools to develop .NET grid applications and grid-enabled legacy applications.

It has been designed with the primary goal of being easy to use without sacrificing power and flexibility. As Alchemi.NET is the emerging technology, so a lot of work has to be done to make it a standard .NET based middleware. We are using Alchemi.NET as the framework to setup the grids.

There can be many reasons for choosing Alchemi.NET as the framework for building the

computational grid and then further choosing to build a fault tolerant system for it. Some of these reasons are:

- Alchemi.NET is the first .NET based stable grid.
- Most important is that Alchemi.NET is open source. So one can do any number of changes in it.
- Another important reason is that most of the systems in our labs are running Windows Operating Systems.
- Fault tolerance research area is still under development in Alchemi.NET.
- Checkpointing is also not available in Alchemi.NET.

3.2 Fault Tolerance in Alchemi.NET

Fault tolerance is the property of a system that helps it to continue operating consistently with its specifications even in the event of failure of some of its parts. From a user's point of view, a distributed application should continue despite failures. Alchemi.NET based distributed system is characterized by a collection of autonomous processing elements, called nodes. Each of these nodes has some computing resources as well as the possibility to exchange information with some of the other nodes. These are referred to as its neighbors and the communication between these nodes take place through a central authority called Manager. The Manager controls the working of all the executors. Users interact with the Manager for submitting and enquiring the status of their jobs.

A ‘grid application’ consists of a number of related grid threads. Grid applications and grid threads are exposed to the application developer as .NET classes / objects via the Alchemi.NET API. When an application written using this API is executed, grid thread objects are submitted to the Alchemi.NET Manager for execution by the grid. Alternatively, file-based jobs (with related jobs comprising a task) can be created using an XML representation to grid-enabled legacy applications for which precompiled executables exist. Jobs can be submitted via Alchemi.NET Console Interface or Cross-Platform Manager web service interface, which in turn convert them into the grid threads before submitting them to the Manager for execution by the grid.

3.2.1 Present Fault Tolerance Scenarios in Alchemi.NET

1. Heartbeat mechanism is used by Alchemi.NET: Executors send the heartbeat messages at some interval to the Manager to whom they are connected. This will help the Manager to maintain the status of the Executors. In case the Manager doesn't receive the heartbeat messages from the executor between the pre-decided times, the Manager consider that executor to be dead and update its information. Here the reason for not receiving of the message is considered to be hardware failure; OS reboot or process being killed. These are termed as hard failure of executor.

2. Executor fails "hard" due to a hardware failure, an OS reboot or the process being killed. In this case Alchemi.NET is using the heartbeat technique to re- schedule the thread to another Executor.

3. Executor fails "soft" due to the user logging off or stopping down the Executor. In this case the Manager is informed that the Executor is going offline and the thread is re-scheduled (this scenario fails many times). This is an acceptable solution.

4. Restart the incomplete job: In case if the job remains unfinished due to any reason like- hardware failure, failure of the Manager, etc the Alchemi.NET Manager stores the status of the job by using "applicatio_id", "internal_thread_id" and "executor_id" fields it maintain in the SQL database.

5. Dynamically determine new nodes added or deleted: As the new node is added or removed, the Manager is being provided with the information dynamically and it updates the console information to make a correct record of the information so as to avoid any faults.

3.3 Problem Formulation

As seen from literature survey Alchemi.NET does not support fault tolerance. Therefore in this thesis a checkpointing algorithm would be implemented for Alchemi.NET.

PROPOSED CHECKPOINTING ALGORITHM 4

This Chapter focuses on the proposed solution. A checkpointing algorithm has been designed for better fault tolerance in Alchemi.NET.Checkpointing is one of the fault-tolerant techniques to restore faults and to restart the job in a shorter time. In this thesis, a new checkpointing algorithm has been proposed which has minimum checkpointing counts equivalent to the periodic checkpointing algorithm, and relatively short rollback distance at faulty situations. The proposed algorithm is better than previously proposed checkpointing algorithms in terms of task completion time in both of fault-free and faulty situations.

In this chapter, Section 4.1 covers the details of proposed checkpoint algorithm for fault tolerance system, which is used in the Alchemi.NET. Section 4.2 covers the basic requirements that are needed for implementation of our work in relation to hardware and software as required. At the end of this chapter, in Section 4.3, we have presented the proposed algorithm and flowchart of Executor and Manager, which are based on the implementation steps.

4.1 Proposed Solution

Checkpointing schemes associate each local checkpoint with a checkpoint sequence number and try to enforce consistency among local checkpoints with the same sequence number. This checkpointing schemes have the easiness, recovery time advantages and low overhead over other checkpointing schemes. Proposed checkpointing algorithm reduces the checkpoint overhead compared to previously suggested checkpointing algorithms and which have a relatively short rollback distance in faulty situation. In the proposed scheme, we use checkpoint sequence number to take a message checkpoint. However, unlike the other algorithms, only one message checkpoint can exist between two consecutive periodic checkpoints. Therefore, our checkpointing algorithm has smaller number of checkpoints than other checkpointing algorithms. Moreover, like the other algorithms, the dependency among checkpoints is removed by message checkpoints. In result, our checkpointing algorithm has a relatively short rollback

distance in faulty situation. The algorithm has a better performance than the others in terms of task completion time in both of fault free and faulty situations. New checkpointing algorithm is discussed here under:

4.1.1 Checkpointing Algorithm

Step 1: The algorithm runs for n processes where the process value P(i) is varies from 0 to n-1.

Step 2: The flag value is initialized. If new checkpoint flag value is (NCF) =0 (False) that means there is no checkpoint flag in the current interval and if the value is NCF =1 (True) that means there is a checkpoint flag in the current interval.

Step 3: After assigning the value of the process, start with while loop when process P(i) till the end.

Step 4: In the checkpoint, if the current time value of the process equals to the checkpoint time than check

```
If (NCF==0)
        Take a stable checkpoint with current process state;
        Increase checkpoint number by 1;
        Checkpoint increase by current time plus checkpoint
        And Set new checkpoint flag = false;
```

Otherwise, if condition does not match take a stable checkpoint with new checkpoint

Step 5: After new checkpoint flag is checked, process P (i) send and receive the message and check

If senders checkpoint number is greater than current checkpoint number than again check

```
 if (NCF==1 )
      Set current  checkpoint number to senders;
 else
   Take a new  checkpoint with current process state;
   Set current  checkpoint number to senders;
    And   Set new checkpoint flag = True;
```

Step 6: At last, process received a message.

Step 7: If any failure occurs in between the execution, after resuming value pick from the memory and go to step 3.

With the new checkpointing algorithm, each process takes a periodic checkpoint at each checkpoint (CP) time. At this point, before taking a checkpoint, each process investigates if there is a new checkpoint in current interval. If one exists, then the content of new checkpoint, which is stored in memory, is written to a stable storage. Otherwise, the current state information of the process is written to a stable storage. After checkpointing, a checkpoint sequence number is increased by 1 and CP time is updated. During normal execution, each process takes a message checkpoint before processing a received message. Each process, before processing message, compares its current checkpoint number with the checkpoint number of message sender that is tagged on each message. If the checkpoint number of message sender is bigger than the current checkpoint number, then each process investigate if there is a new checkpoint in current checkpoint interval. If a new checkpoint is exists, then set the current checkpoint number to the message sender's checkpoint number and process the received message. Otherwise, the current state information of process is written *to* memory (new checkpoint) and current Checkpoint number is set to the message sender's checkpoint number. After taking new checkpoint, each process executes the received message.

4.1.2 Complexity Analysis of the Proposed Algorithm

The Complexity of the above algorithm is O(n). Because n processes run n time within while loop. For each process P(i), the if-else statements with in the while loop gets executed. Assuming that first if-else loop runs j times, where j is a constant when the current time equals to the checkpoint time. After that the process P(i) has to process the received message if sender checkpoint number is greater than the current checkpoint. The if-else construct runs k times, where k is a constant and since the algorithm is running for the n processes so the total complexity of the algorithm is O(j+k)*n. which is approximately equal to the O(n). The O(n) complexity of the algorithm is better to implement any type of checkpointing algorithm in real time environment. For the requirement for the implementation of the new framework is given below.

4.2 Implementation Requirements

The required configuration for the implementation of my proposed work is

4.2.1 Hardware Requirements

The minimum configuration for hardware is:

1. Personal computers – PIII, PIV
2. Local Area Networks

4.2.2 Software Requirements

The minimum configuration for software is:

1. Microsoft Windows 9x/NT/2000/XP/2003
2. Microsoft .Net framework 1.1
3. Microsoft SQL Server 2000 or MSDE 2000 installed with security
4. ASP.NET
5. Internet Information Services (IIS)
6. Alchemi.NET.NET 1.03
7. Visual Studio .net development kit

4.2.3 Programming Language

Our choice of programming language depended on several factors:

- Platform Independence,
- Object Serialization Support,
- Multithreaded Program Support, and
- GUI support, with consistent user experience (look and feel) across different platforms.

As C# satisfies all these factors, it is the best match for implementing the proposed work in this thesis.

4.3 Implementation of Executor Algorithm

The flowchart in the figure 4.1 shows the steps of Executor.

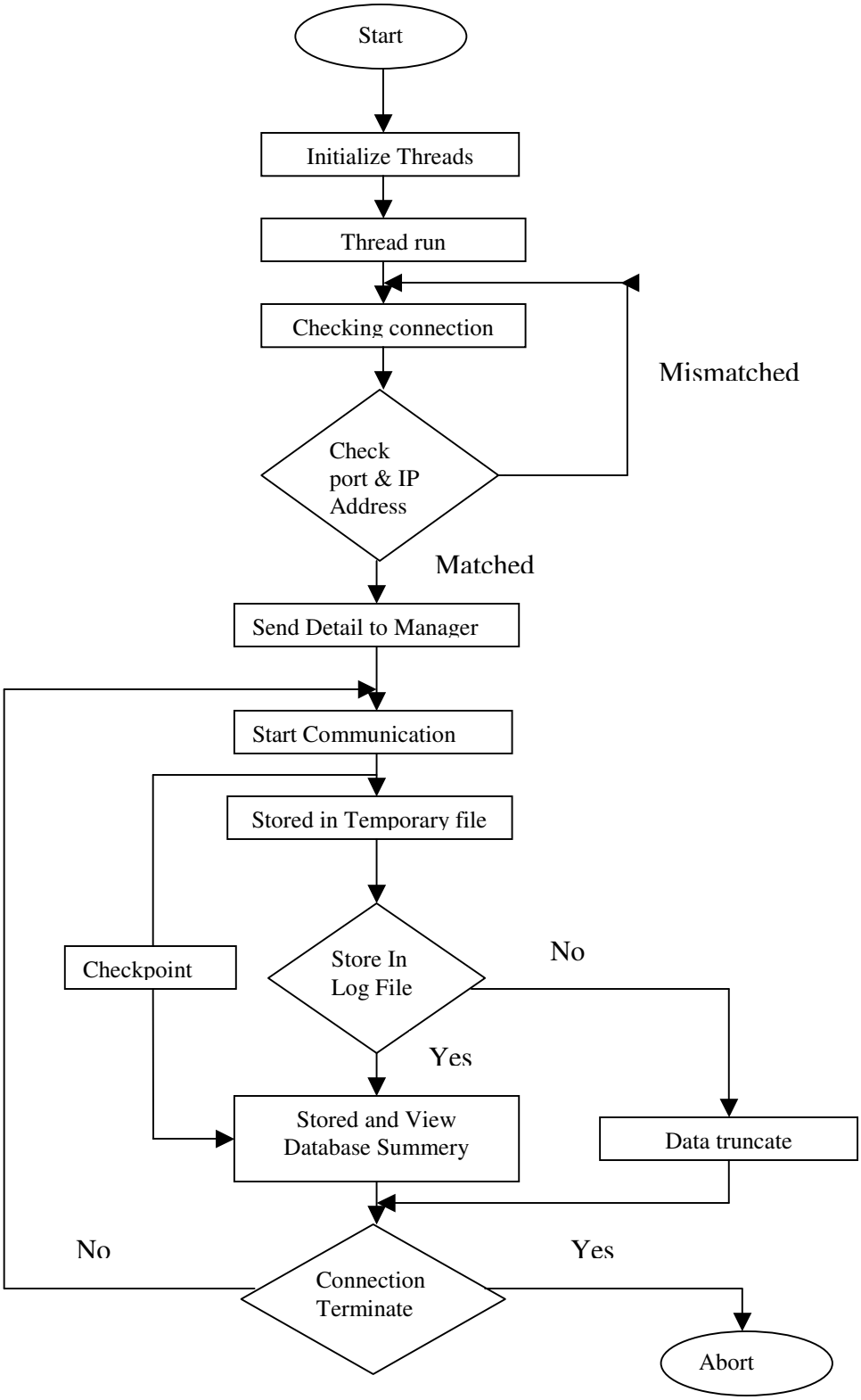


Figure 4.1 Flow Chart of Executor

Steps of Implementation for Executor Algorithm

Step 1: Start the Executor and import all the threads like System.Net.Sockets, System.Threading, and System.IO.

Step 2: Initialize variables of Binary Reader and Writer, thread, TcpClient NetworkStream.

Step 3: Start the Executor thread. Clint_thrd1. Start ()

Step 4: Assign port number to socket and wait for the connection on the basis of same port number and connecting Manager IP Address.

tcp_client.Connect ("local host", 45)

Step 5: if (new_connection connected=True) then

Connection established and sends Executor detail to Manager

Else Connection Fail. Again check for new connection

End if

Step 6: After connection establish data Sending or receiving through the stream writer and stream reader and display the result through the message box.

reader = New BinaryReader (net_stream)

writer = New BinaryWriter (net_stream)

Step 7: All the communication data is parallel stored in the temporary file and display the result at both the Manager and the Executor site or after a specific time of checkpoint data will save at the permanent storage

If (e.KeyCode = Keys. Enter) Then

writer. Write (inputttxt.Text)

outputtxt.Text &= "Executor side:-" & inputtxt.Text & vbCr

RichTextBox2.Text += vbCrLf & "Executor side:-" & inputtxt.Text & vbCrLf

End If

Step 8: If want to store data in the log file then saved data otherwise truncate data and terminate theconnection.RichTextBox1.

LoadFile("d:\file", RichTextBoxStreamType.PlainText)

Step 9: We can see the hole save data record through the database summery record.

Databse_summery.Show ()

Step 10: After this if u want to new entry connection then go to entry detail otherwise abort the connection. tcp_client.Close () and clint_thrd1.Abort ()

4.4 Implementation of Manager Algorithm

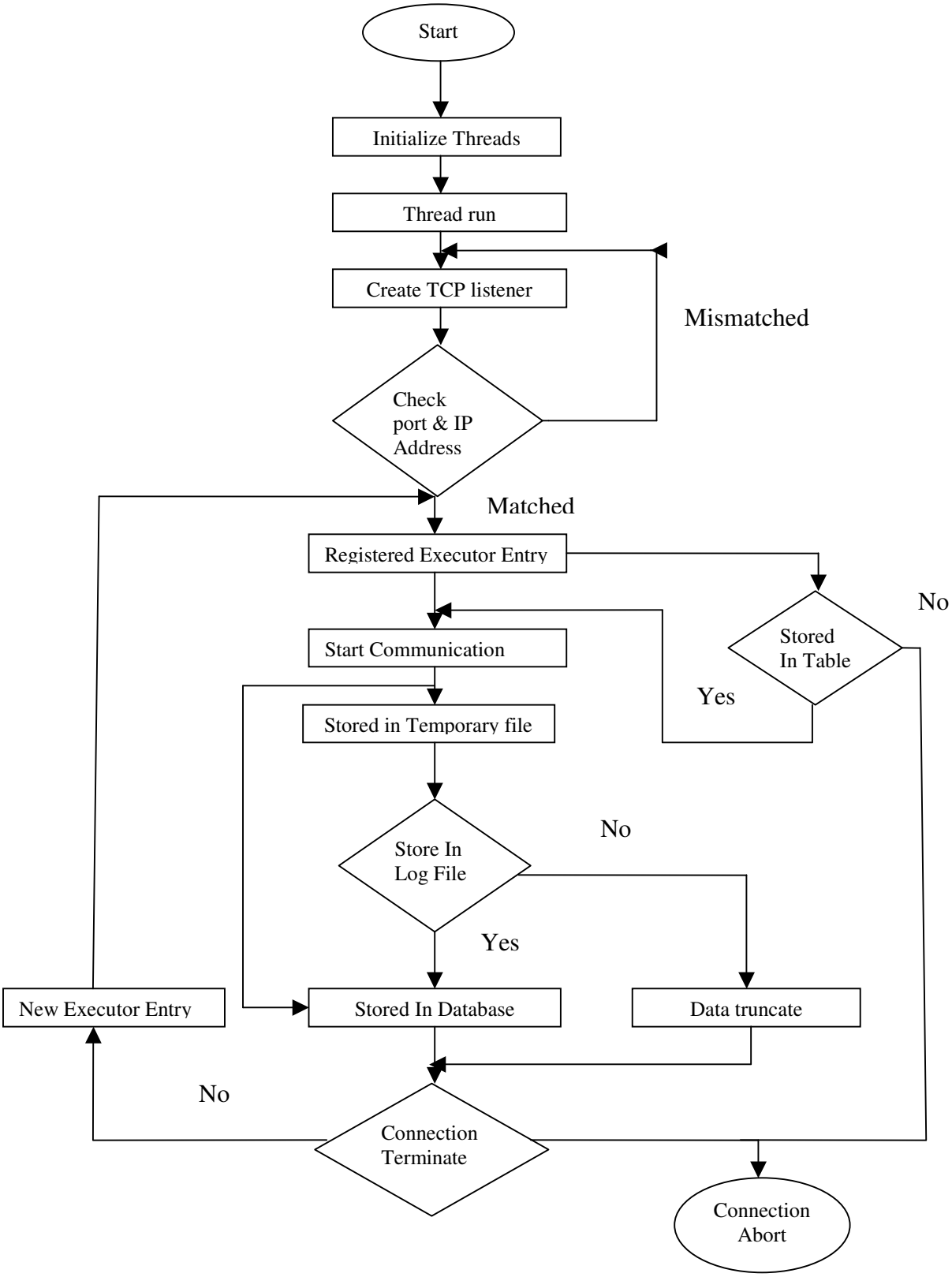


Figure 4.2: Flow Chart of Manager

Above the flowchart in the figure 4.2 shows the steps of Manager.

Steps of Implementation for Executor Algorithm

Step 1: Start the Manager and import System.Net.Sockets, System.Threading, System.IO threads.

Step 2: Initialize variables of BinaryReader, BinaryWriter, thread, sockets and new_ connection, NetworkStream, Net.Sockets.TcpListener.

Step 3: Run the Executor ser_thrd1.Start () thread.

Step 4: Create tcp listener on the basis of assign port number to socket and wait for the connection to new Executor.

```
tcp_listn = New Net.Sockets.TcpListener  (System.Net.IPAddress.Any, 45)
tcp_listn.Start ()
```

Step 5: Establish new connection between Manager and Executor through TCP listener socket and

```
If (new_connection.connected=True) then
        Connection established
Else
        Connection Fail. Again check for new connection
End if
```

Step 6: After establish the connection registered the Executor into the Executor table in Alchemi.NET.NET database and start communication otherwise abort the connection.

Step 7: Sending and receiving (read/write) data through the stream writer and stream reader and after the connection establish display the result through the message box.

```
writer = New BinaryWriter (net_stream)
reader = New BinaryReader (net_stream)
outputtxt.Text &= "Connection Established" & vbCr
Dim recive_data As String
   Do
       recive_data = reader.ReadString
       outputtxt.Text &= "Executor side:-" & recive_data & vbCr
   Loop While (recive_data <> "terminate")
```

Step 8: All the communication data is parallel stored in the temporary file and display the result at both the Manager and the Executor site or after a specific time of checkpoint data will automatically saved in the permanent storage

```
If (e.KeyCode = Keys.Enter) Then
   writer. Write (inputtxt.Text)
   outputtxt.Text &= "Manager Side:-" & inputtxt.Text & vbCr
End If
```

Step 9: If want to store data in the log file then saved data otherwise truncate data and terminate the connection.

Step 10: After this if u want to new entry connection then go to entry detail otherwise abort the connection.

```
tcp_listn.Stop ()
ser_thrd1.Abort ()
```

4.5 Summary

In this chapter, we have proposed the new communication induced checkpointing algorithm which is use to make Alchemi.NET fault tolerant. We have also described the need of hardware and software requirement in the implementation, programming language which is use for our implementation work. After that we have discussed the proposed and designed framework and described the algorithm and flowchart, which is done, on the Alchemi.NET middleware.

The next chapter presents the screen shots and database details of our research work.

Experimental Results 5

This chapter describes the implementation of Checkpointing algorithm, in which data can be stored in permanent log file on the basis of checkpointing, which has been proposed in the previous chapter. An application developed in VB.Net, C# (front end) and SQL SERVER (back end database). The results are shown in the form of screen shots.

Alchemi.NET grid can be viewed as a virtual machine with multiple processors. A grid Application can take advantage of this by creating independent units of work to be executed in parallel on the grid (each unit of work is executed by a particular Executor).

These units of work are called grid threads and must be instances of a class that is derived from Alchemi.NET.Core.Owner, GThread. Code that is to be executed on the grid is defined in this class's void Start () method.

5.1 Implementation of Checkpoint Scheme

Follow these steps to set up a development environment:

- Construct a minimal grid (1 Manager and 1 Executor) on the development and test it by running Application.
- Download the Alchemi.NET SDK and extract to a convenient location
- Locate Alchemi.NETCore.dll for referencing in applications.

Checkpointing is implemented using the below mentioned steps:

Step1: In this kind of application first check the status of the Manager then, start application to scan port 9001 of the remote system where Manager is running at regular interval of time.

The code for this application is written in c#. The two main classes used for this are:

'Socket' and 'IPEndPoint'. User has to provide the IP address of the system where Manager is running. In any such case when the Manager fails due to some reason, data is saved on the log tables. After recovering, it can again start from the checkpoint created.

Step2: Creating the checkpoint Database

Checkpoint is the process of stored data in regular time intervals from memory to permanent storage databases. Using checkpoint, small copies of data stored by us into the

database can be shared by both Manager and Executor. The previous data records are safely stored in log file, which is present on the executor site. The database structure is replicated in the following table. These tables maintain all the information regarding executor, application and thread etc. that is presented in the screen shots below.

Table 5.1: Detail of running threads

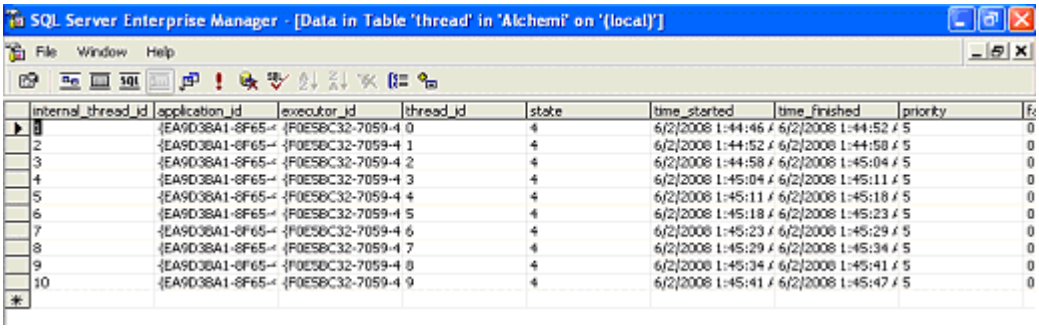
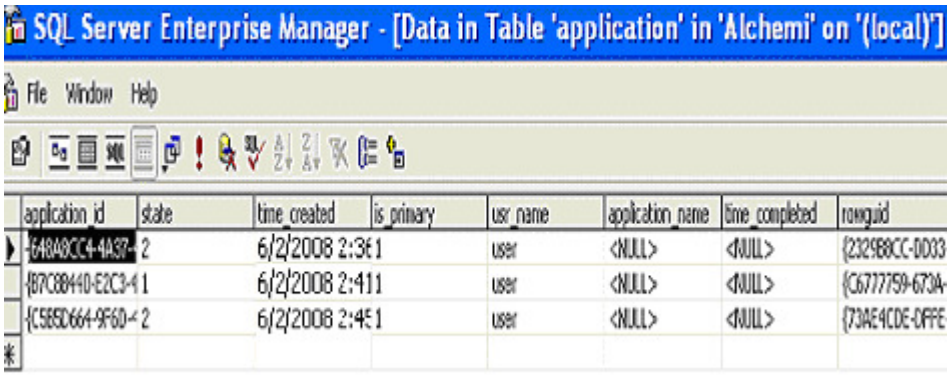


Table 5.2: List of running application



Steps for Open database tables in the SQL server:

1. Open SQL Server Enterprise Manager and select Tools menu ->Console root -> Microsoft SQL server--> SQL Server Group.
2. Configure the appropriate Server Service Manager and set server and services.

3. Enable the appropriate database for merge replication.
4. Enable the appropriate server as subscriber.
5. Start the status of the SQLSERVERAGENT service as running. Just click Next.
6. Open SQL Server Enterprise Manager and select the appropriate SQL Server Group for database entry.
7. Choose the Alchemi.NET database to be opened and Click Next.
8. Specify the Subscriber Types. Select "Servers running SQL Server 2000". Then click Next.
9. Select the Object Types (name of tables present in the Alchemi.NETdatabase)
10. Which you want to open, and click Next.
11. Open appropriate table, which want to open and check all entry through click, return all the rows. It will merge the necessary data. Refresh it once.

5.2 Experimental Evaluation

After successfully building the checkpointing algorithm, it has been tested in three stages. First the test strategy has been formed and then the test cases has been defined and finally the results are presented.

5.2.1 Test Strategy

In the implementation, we have taken a part of different departments of the Thapar University in as computational grids. For building the application for our system, we have used C# as programming language and MS SQL Server as the database. The checkpoint Manager for managing the proper working of Manager is implemented using the steps described above and tested our system by running the sample application.

Test Cases

Following are the test cases that we are going to use to check our system.

Test Case #1 Executor On/Off

We dynamically switch on and off the executors. While doing this, we kept on checking the status of the executing jobs through the Alchemi.NET Console Manager.

Test Case #2 Checkpointing between Manager and executor

We have been checking the working of the Manager. This module is just like a checkpoint monitoring system, in which the Manager invokes checkpoint signal at a regular interval of time

Test Case #3 Automatic Checkpointing

We check this case with the help of automatic checkpoint monitor facility provided in the MS SQL server. It uses the snapshot agents to update the database and present information.

Test Case #4 Manager Stopped

This case is tested in two parts. First, by manually disconnecting the Manager and noting down the results and secondly, by switching off the system without disconnecting the Manager.

5.2.2 Experimental Results

While testing and checking many possibilities of fault occurrence first check the four cases which are discussed above. We found that our proposed system helps to increase the performance of grid by providing the checkpointing concept. When the system fails, the checkpoint saved data successfully comes into foreplay. The test case three was tested using the Microsoft SQL Server built in facility to monitor. Test case 4 is the main test case that was to be tested. Executors have to be reconnected to this checkpoint.

Figure 5.1 shows the interface of the Alchemi.NET executor .it has three button first connect button for establish the connection between Manager and the executor, second disconnect button for terminate the connection between them and the last one database summery for shown the all communication which was held between both. It can show the permanent log file data that are stored in the disk

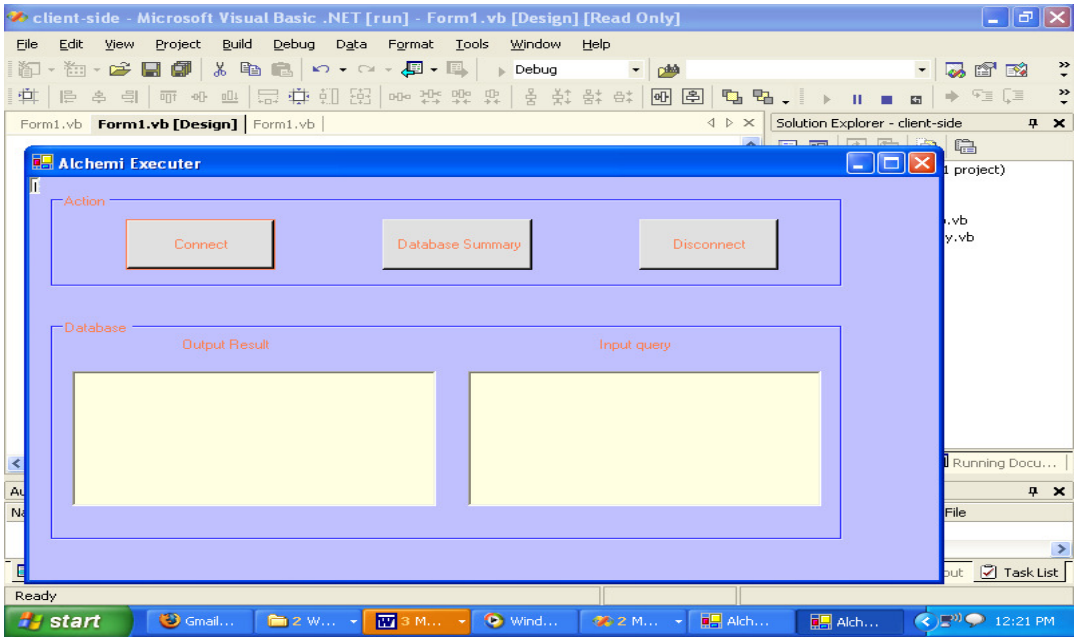


Figure 5.1: Executor user interface

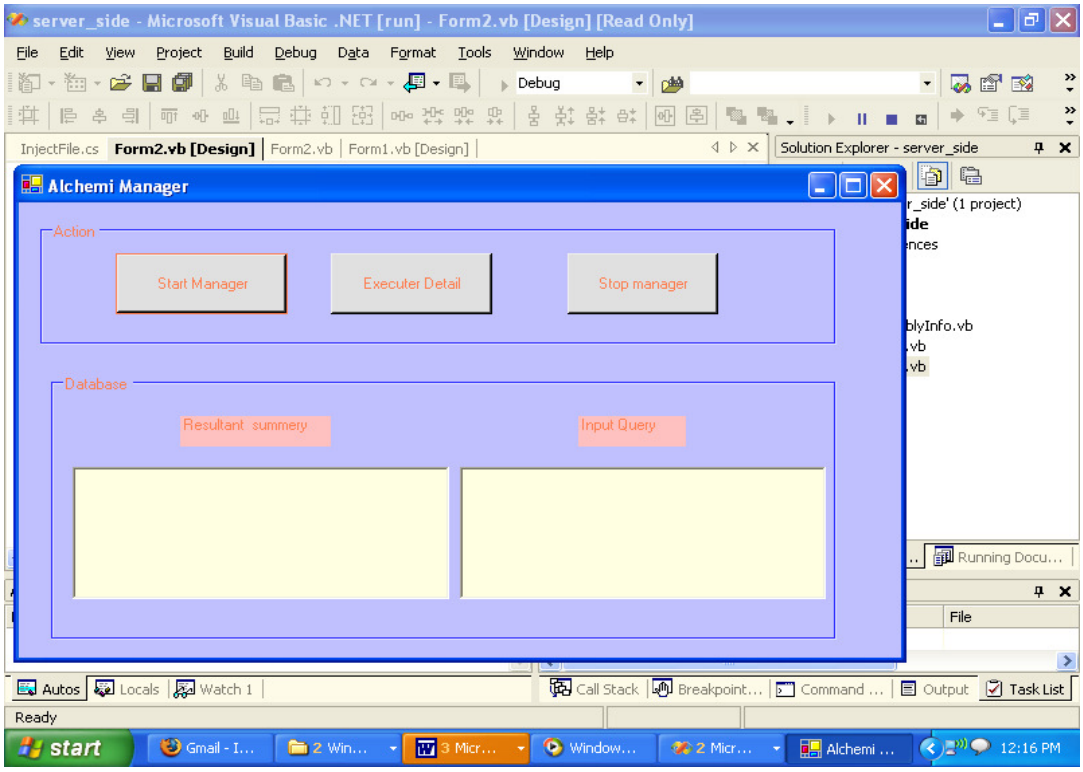


Figure 5.2: User Interface of Alchemi.NET Manager

Above image shows the interface of the Alchemi.NET Manager. It has three buttons, first is the Start Manager button, for starting the Manager. Second is the stop button for terminating the connection between them and third button for displaying new form in

which we can enter the executor details to store in the executor table. It also has two rich text box, one for input query in which we can communicate to executor and other one shown is the resultant summary which shows all communication which was held between both. It saves data on temporary basis which is in the memory.

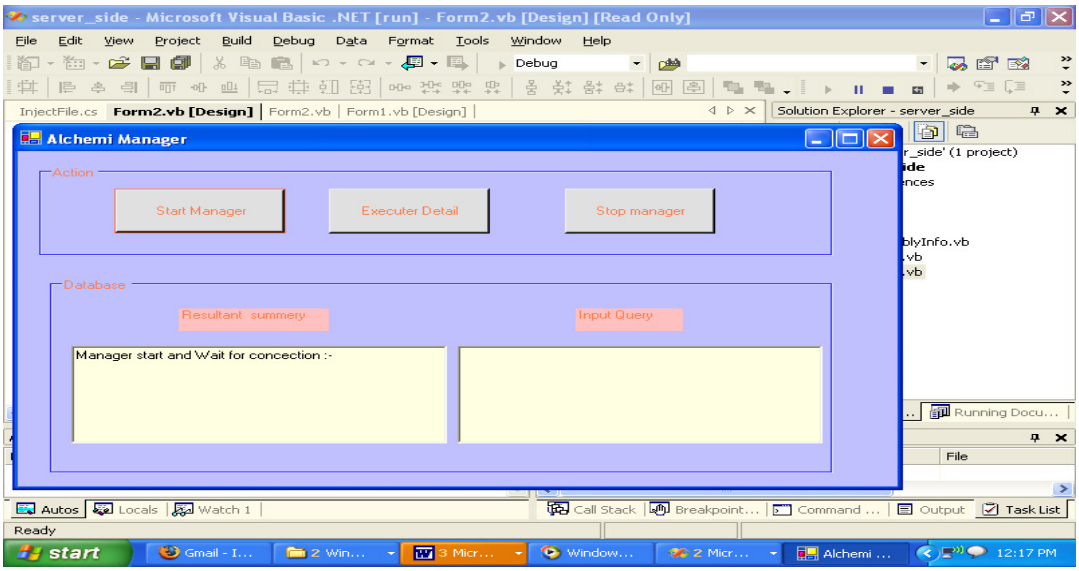


Figure 5.3: After Alchemi.NET Manager start and wait for Connection

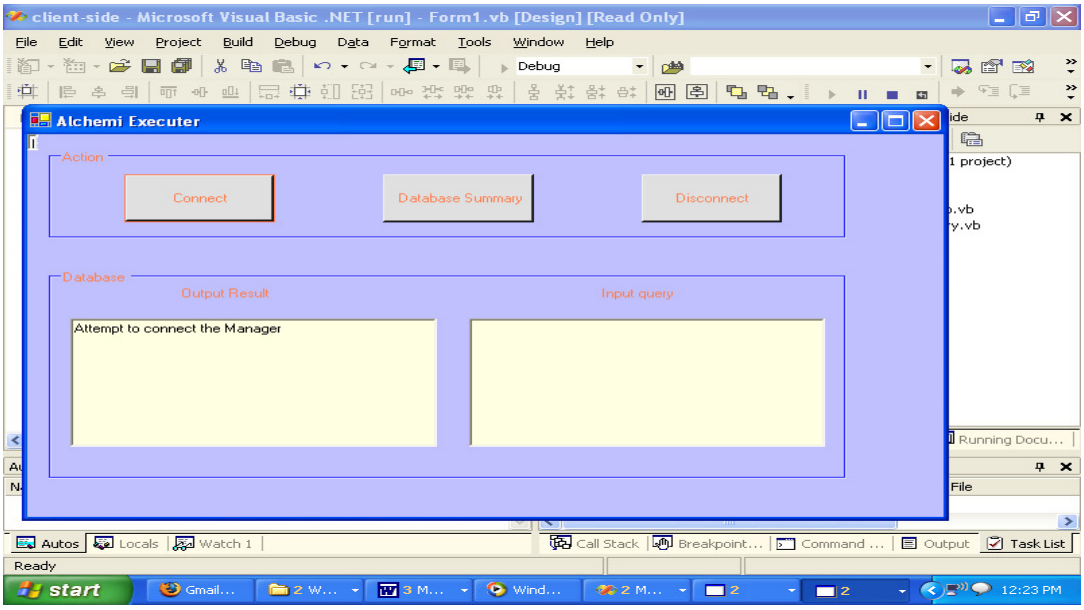


Figure: 5.4 After Alchemi.NET executor Start and wait for Connection

Above images shows that the Alchemi.NET Manager and Executor has started and they are further waiting for the heartbeat signal for establishing the connection with each other for further processing.

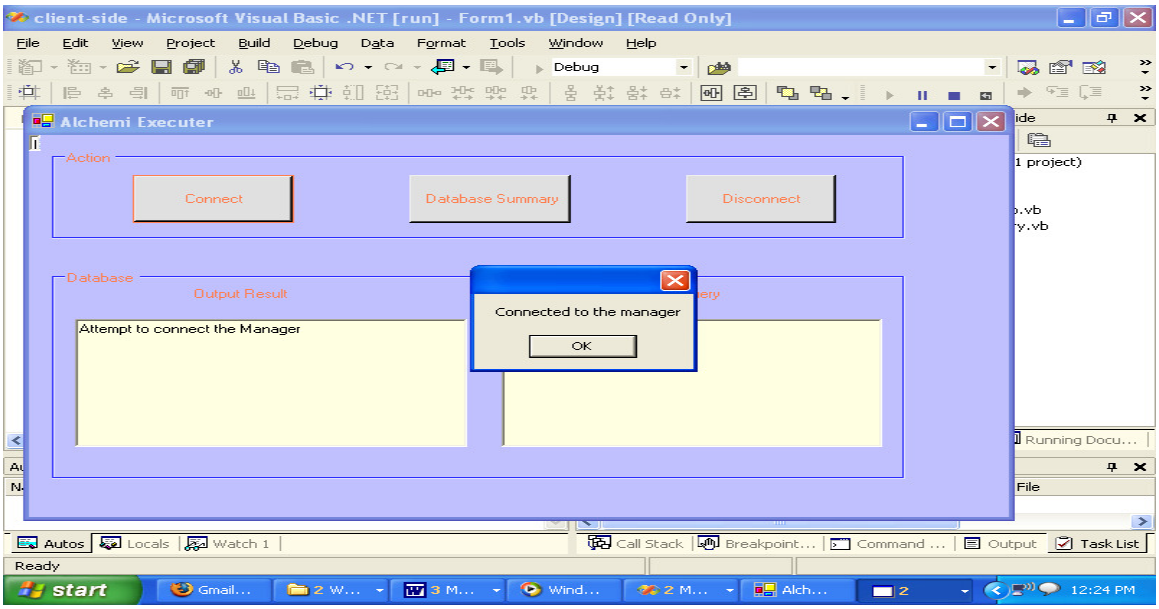


Figure: 5.5 Alchemi.NET Executor connect to Manager

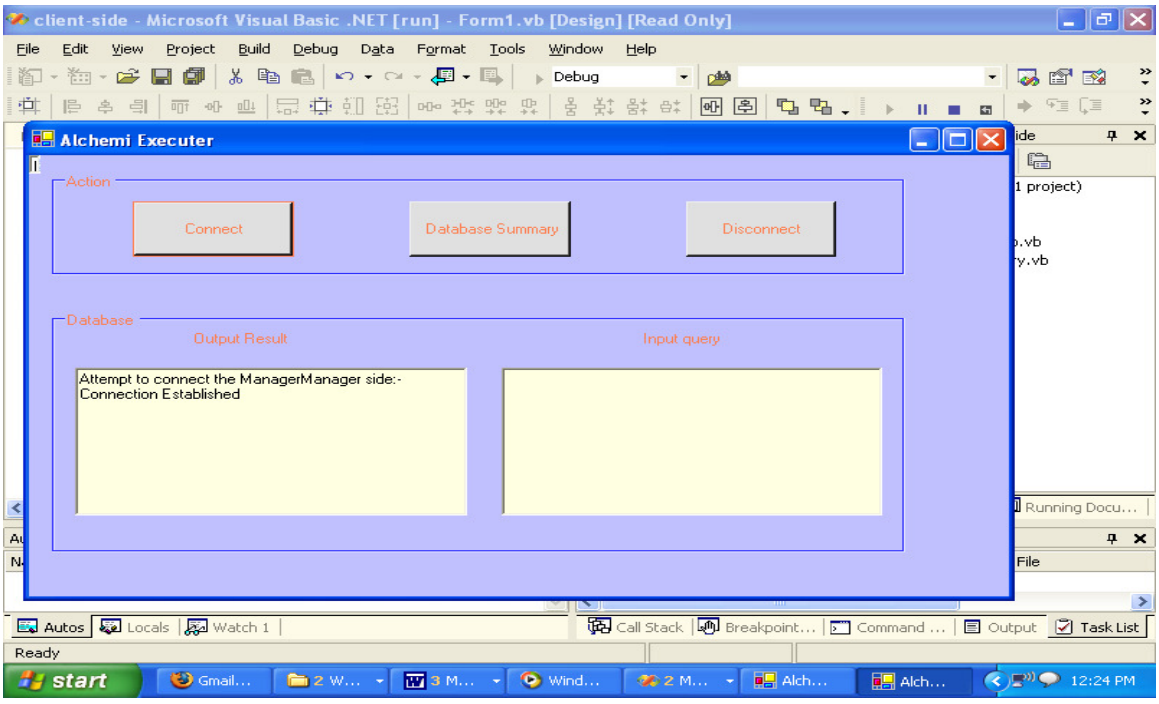


Figure: 5.6 After establish the connection to Manager

The above images shows the established connection at the Manager and further acknowledge should be displayed on the Executor site.

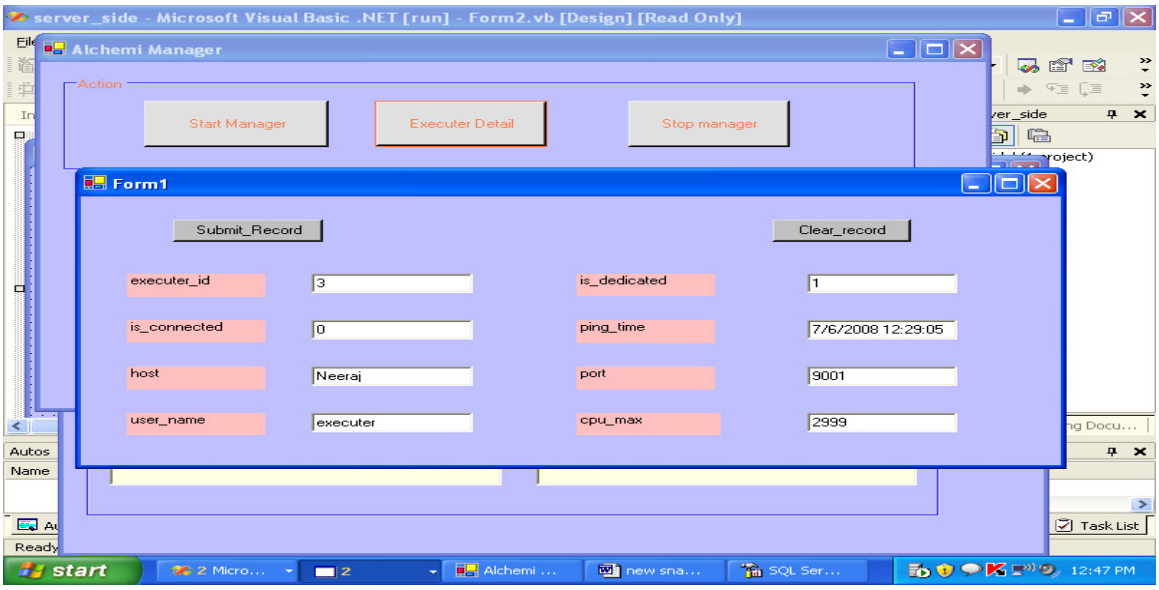


Figure: 5.7 Filled Alchemi.NET executor entry form

Above image shows the entry form of the particular executor, which is currently connected, to the Manager. It has all the entries filled according to executor requirement. There are two buttons, one for submitting the entry in the table and other for the clearing entry if any field is filled incorrectly. After filling the entry in executor form all the data is stored in the executor database table in the Alchemi.NET database.

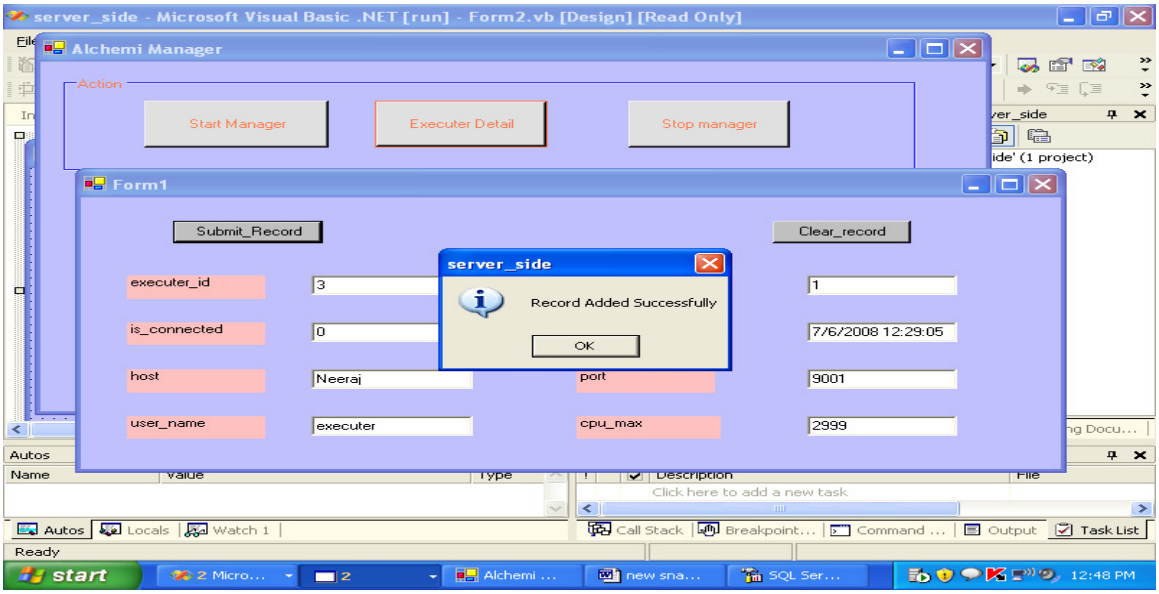


Figure: 5.8 After Successfully storing data in the table

Figure 5.8 shows the successful record added in the executor table. This is permanent storage of data entry.

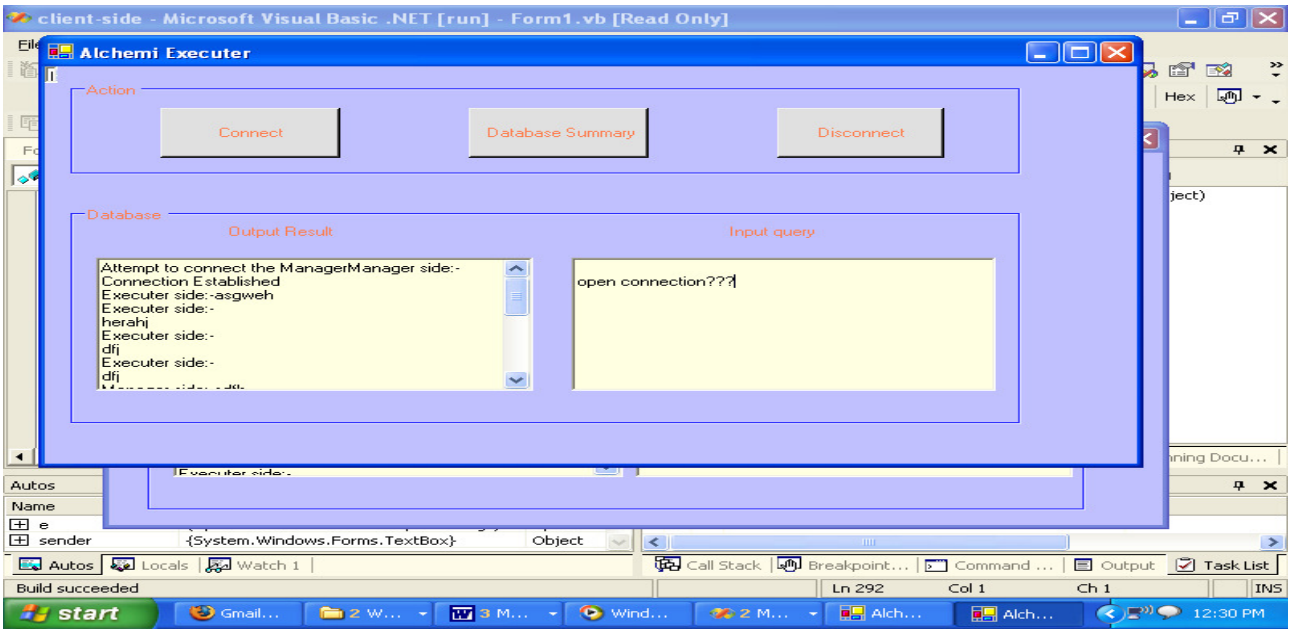


Figure 5.9 Communication at executor site

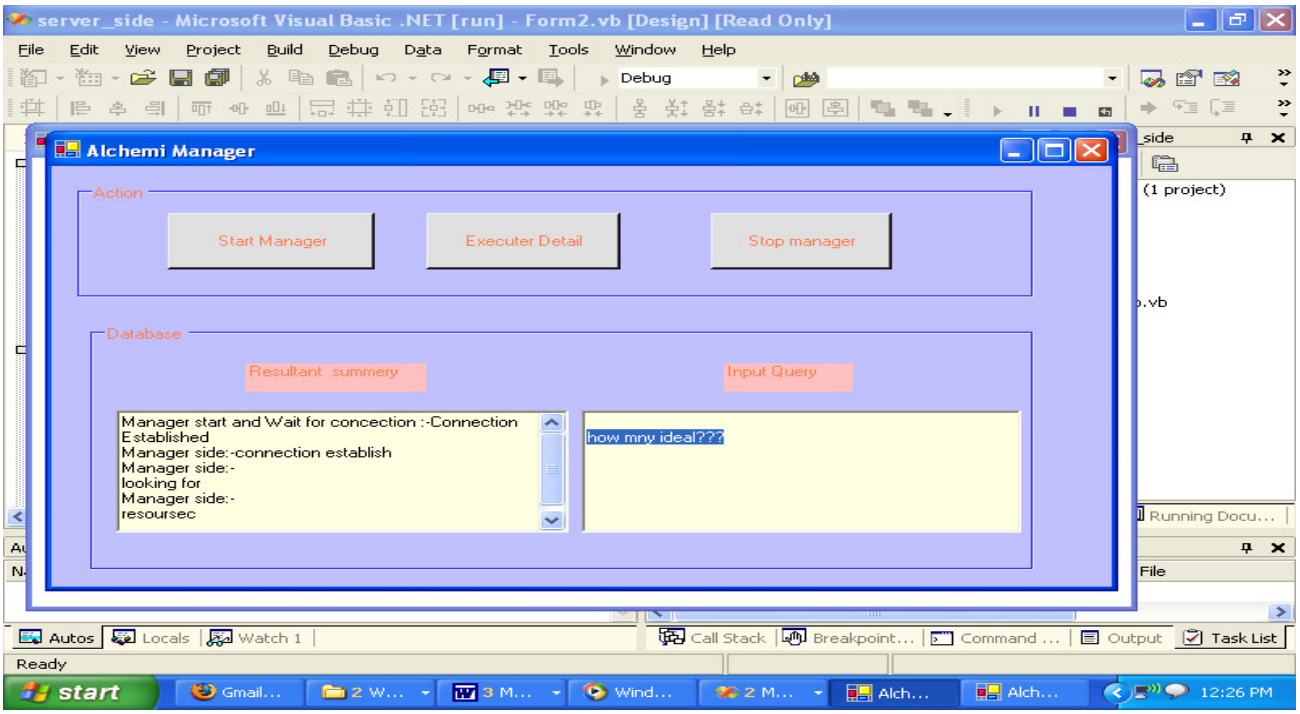


Figure: 5.10 Communication at Manager site

Above Figures, shows how to send messages between Manager and executor. First send the message from the input query and temporary store all data shown at output result.

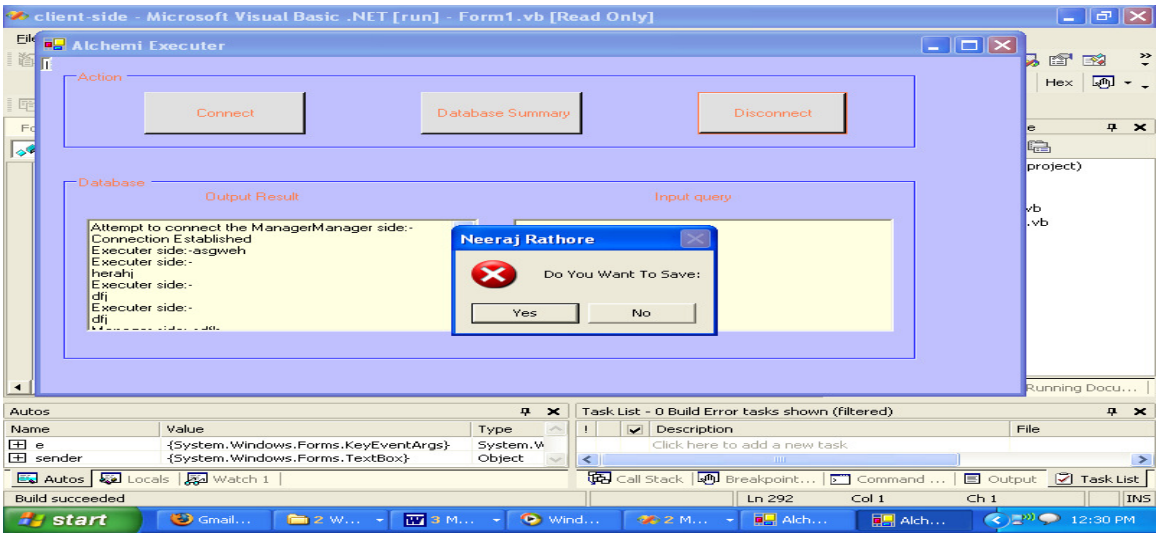


Figure: 5.11 Shows the message to store or delete the data

Figure: 5.11 show the message box Yes for saving all the communication in permanent log file and No for truncating data between Manager and executor.

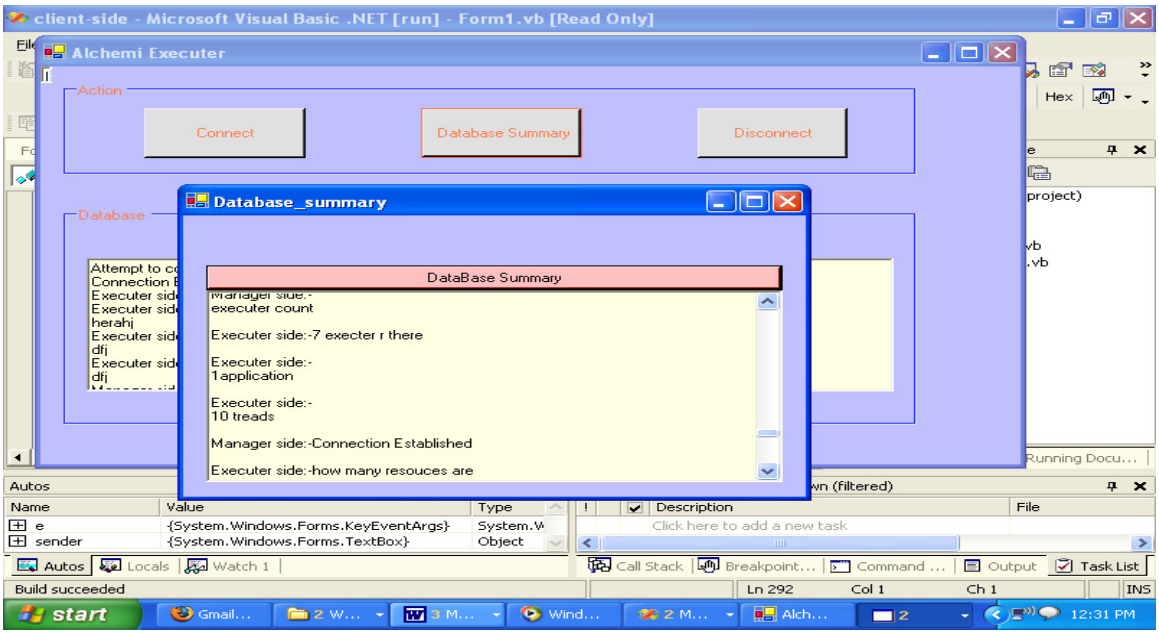


Figure: 5.12 Shows the database summery

Figure: 5.12 shows the database summary form in which we can see all the log file data at Manager Site, which is permanently stored.

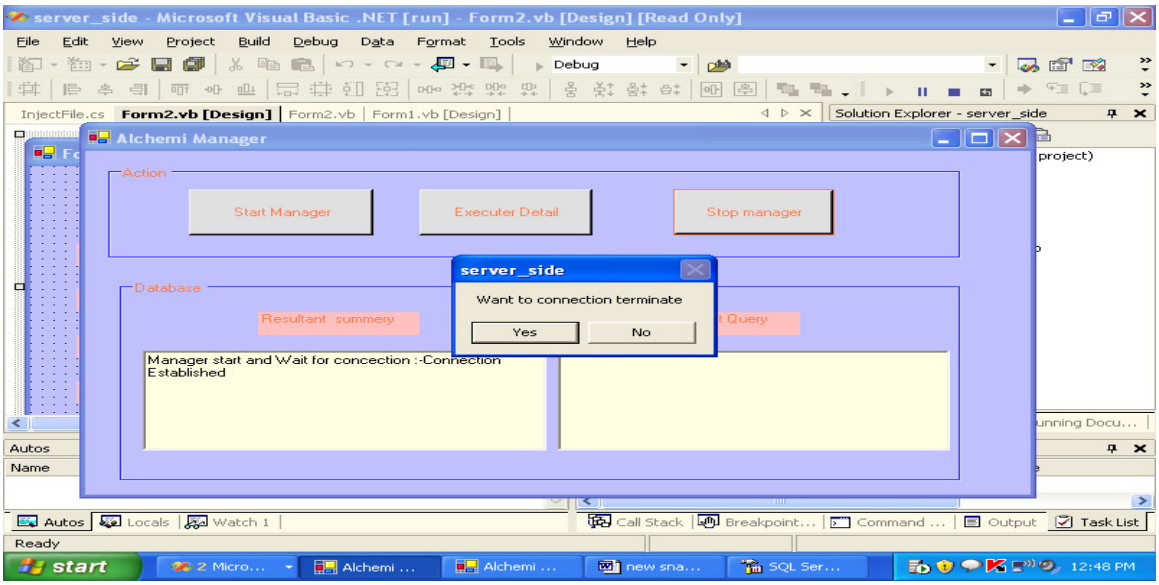


Figure: 5.13 Shows the termination of connection to executor

Figure: 5.13 shows the message box, Yes for terminating the connection at the Manager site and No for further processing.

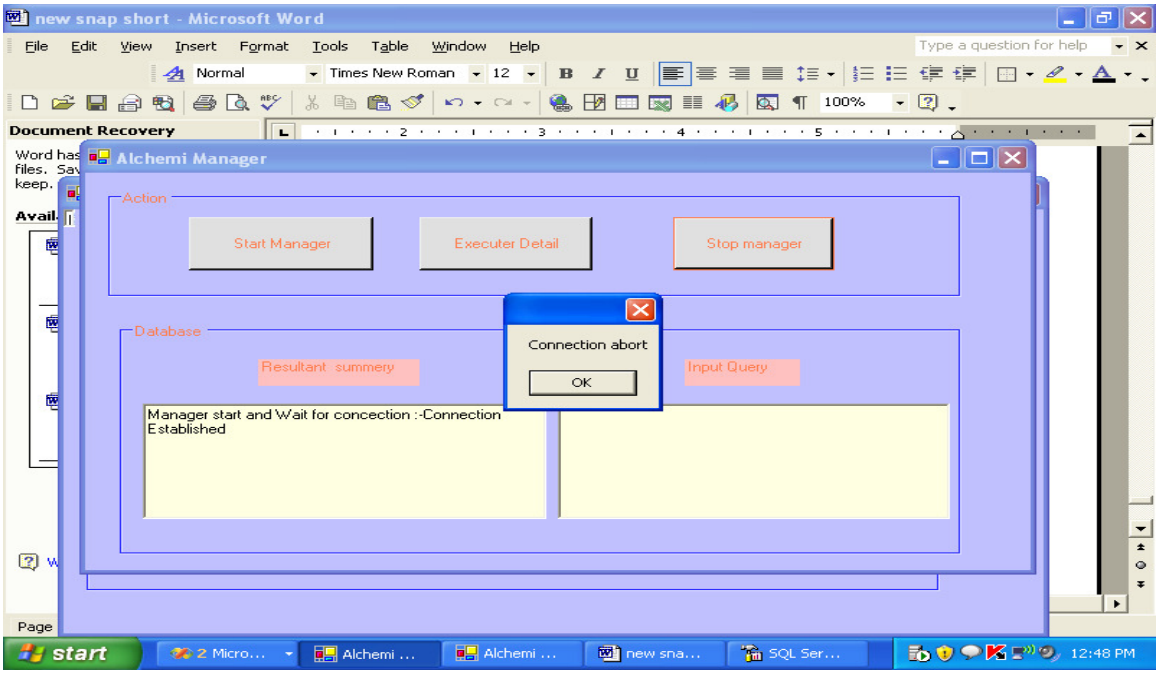


Figure: 5.14 Shows the terminate all the link between executor and Manager

Figure: 5.14 show the message box after clicking Yes in previous form and after pressing OK on the message box, above connection is aborted.

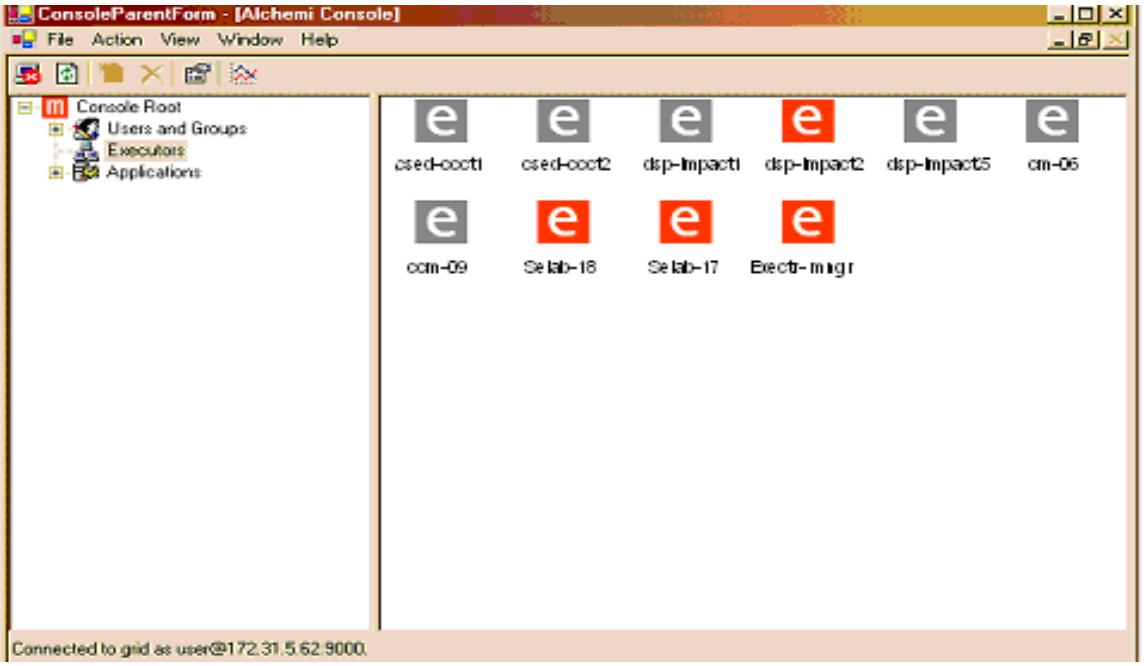


Figure 5.15: shows Four Executors running while executing application

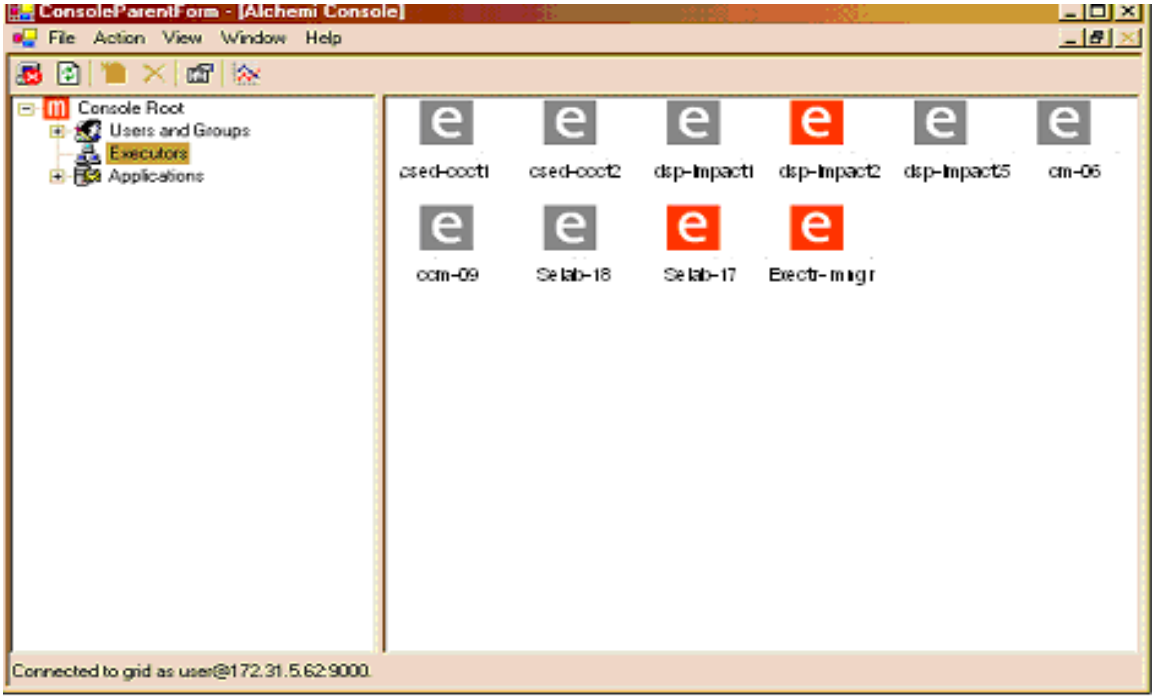
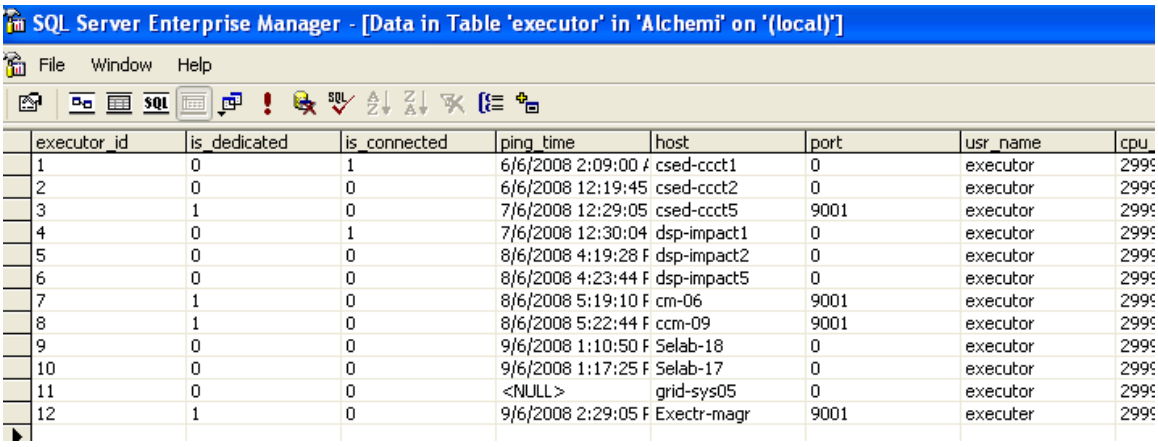


Figure 5.16: One-Executor stops while the application is running

Above figures shows the on line Executor who are connected to the Manager while running the application. Some of the Executor who are red in the figure are ideal and ready to execute the job/application and other gray in colour Executor shows that they are not ideal at the time of any application running.

Table: 5.5 Executor table after storing executor data



executor_id	is_dedicated	is_connected	ping_time	host	port	usr_name	cpu_
1	0	1	6/6/2008 2:09:00 A	csed-ccct1	0	executor	2999
2	0	0	6/6/2008 12:19:45	csed-ccct2	0	executor	2999
3	1	0	7/6/2008 12:29:05	csed-ccct5	9001	executor	2999
4	0	1	7/6/2008 12:30:04	dsp-impact1	0	executor	2999
5	0	0	8/6/2008 4:19:28 P	dsp-impact2	0	executor	2999
6	0	0	8/6/2008 4:23:44 P	dsp-impact5	0	executor	2999
7	1	0	8/6/2008 5:19:10 P	cm-06	9001	executor	2999
8	1	0	8/6/2008 5:22:44 P	ccm-09	9001	executor	2999
9	0	0	9/6/2008 1:10:50 P	Selab-18	0	executor	2999
10	0	0	9/6/2008 1:17:25 P	Selab-17	0	executor	2999
11	0	0	<NULL>	grid-sys05	0	executor	2999
12	1	0	9/6/2008 2:29:05 P	Exectr-magr	9001	executer	2999

Table 5.5 shows the entire executor list, which were connected to the Manager. The table, has all the record about the executor with the entire field that is used by the executor.

5.3 Summary

This chapter deals with the implementation part of thesis work. After giving the basic requirements for implementation, we have proposed the system that deals with the problems found in Alchemi.NET.NET based computational grids has been proposed. The system is implemented using C# as programming language and MS SQL Server as Database. Considering different test cases the experimental evaluation was also done for the implemented system.

In the next chapter concludes our thesis work and gives future scope of it.

CONCLUSION AND FUTURE WORK 6

The sharing of computational resources is a main motivation for constructing Computational Grids in which multiple computers are connected by a communication network. Due to the instability of grids, the fault detection and fault recovery is a very critical task that must be addressed. The need for fault tolerance increases as the number of processors and the duration of computation increases.

In this thesis, we investigated the issues and challenges involved in dealing with faults in Alchemi.NET middleware have been investigated and the checkpoint algorithm in Alchemi.NET for dealing with various kinds of faults has been implemented. Further the efficiency of our proposed system under various conditions has been evaluated.

This chapter provides some concluding remarks, highlights the main contribution of this thesis and presents some possible future research directions. This chapter is organized as follows: Section 6.1 summarizes the project, section 6.2 presents the main contribution of this thesis, and finally section 6.3 highlights the several possible future research directions.

6.1 SUMMARY

Advances in networking technology and computational infrastructure make it possible to construct large-scale high performance distributed computing environments that provide dependable, consistent and pervasive access to high end computational and heterogeneous resources despite geographical distribution of both resources and users. In a heterogeneous computing environment like Grid, a suite of different machines ranging from personal computer to supercomputer is loosely inter-connected to provide a variety of computational capabilities to execute collections of application tasks that have diverse requirements. These kinds of large scale high performance distributed services have recently received substantial interest from both research as well as industrial point of view. However, an important research problem for such systems is the lack of fault tolerance system. The heterogeneous and dynamic nature of grids makes them more prone to failures than traditional computational platforms. So, the system managing such infrastructures needs to be smart and efficient to overcome the

challenge of fault detection and recovery.

The first chapter of this thesis introduces grid computing, discusses about the history, types, and the motivation of this research work. Different ways to handle these failures architecture, concept of virtual organization and faults in grid are discussed in the literature review part of the thesis. After studying fault tolerance in different grid middleware Alchemi.NET has been chosen, existing the reasons for which have been discussed in chapter 3 with some present scenarios and the brief introduction of the problem formulation. After analyzing the detailed performance of Alchemi.NET have been identified. Different areas related to fault tolerance in which the discussed middleware still lacks. It has been realized that, in order to support fault tolerance in computational grid (Alchemi.NET Framework based), it is useful to provide a checkpointing support for avoiding the grid to be down during failures. The chapter 4 covers the details of proposed checkpoint algorithm for fault tolerance system, which is used in the Alchemi.NET and the basic requirements that are needed for implementation of provides the proposed algorithms and flowcharts of Executor and Manager, which are based on the implementation steps. At last, experimental results and screen shots of proposed framework have been shown in chapter 5.

6.2 MAIN CONTRIBUTIONS

- Studied the Fault Tolerance in different grid middleware's and found most common kind of failures that can let the Alchemi.NET grid to work improperly..
- Alchemi.NET based grid has been set up in the departments.
- Various existing fault tolerance mechanisms already present in the setup grid are identified. After studying the internals of Alchemi.NET, different shortcomings that are there in Alchemi.NET for handling the faults dynamically has been identified.
- On the basis of identified shortcomings, proposed and updated fault tolerance in Alchemi.NET middleware with the help of checkpointing algorithm
- Implemented checkpoint concept that will help in avoiding the grid to become inaccessible if the Manager or Executor fails.

6.3 FUTURE WORK

The future extension of the project can be done in three ways:

Extension of the thesis implementation

- The current implementation is done within LAN with limited number of PCs, this can be extended for systems outside the city network.
- If the failure occurs, then the backup is stored in log file. But this is to be done manually. There must be some provision so that on failure of Manager executors will automatically store data in log file.
- This implementation is kind of third party software that will improve the grid fault tolerance. But this can be integrated into the Alchemi.NET middleware code to help the user to easily use it.

Extension of proposed method

- Our approach of checkpoint can be extended to work on support for multiclustering with peer-to-peer communication between Managers.
- Execution thread hanging problem must be fixed. Two ideas to handle this problem are: One idea is to monitor the executing thread and terminate it if it exceeds a configurable amount of time. This value should be configurable from the application so the user can set it but an override at the Executor level is probably desirable as well. One problem here is that some machines take longer to execute something than others so maybe if the time it waits is a factor of the computer's speed it might be useful. Another idea I have been toying with is to require long running threads to raise status events containing a "percent done" value. The monitoring thread would then have data to see if the thread is dead or just taking longer to complete but still alive. The events could have enough information to be used as a checkpoints but this would be up to the implementation of each application.

APPENDIX A

Setting up Computational Grid: Alchemi.NET

We can define a "computational grid" as being any distributed computational network, enabled with software that allows cooperation and the sharing of resources. Computational grids can be developed using many middleware. We are here using the Alchemi.NET: A .NET based middleware for establishing the computational grid environment. The steps necessary to realize a computational grid include:

- The integration of individual software and hardware components into a combined networked resource.
- The implementation of middleware to provide a transparent view of the resources available.
- The development of tools that allows management and control of grid applications and infrastructure.
- The development and optimization of distributed applications to take advantage of the resources.

Steps for Installing Computational Grid: Alchemi.NET

These are the steps for installing various Alchemi.NET Components. For every component we are giving its installation, configuration and operations are defined.

1. **Alchemi.NET Manager**

Installation

The Alchemi.NET Manager can be installed in two modes

- As a normal Windows desktop application
- As a windows service. (Supported only on Windows NT/2000/XP/2003)

- To install the Manager as a windows application, use the Manager Setup installer. For service mode installation use the Manager Service Setup. The configuration steps are the same for both modes. In case of the service-mode, the “Alchemi.NET Manager Service” installed and configured to run automatically on Windows start-up. After installation, the

standard Windows service control Manager can be used to control the service. Alternatively the Alchemi.NET Manager Service Controller program can be used. The Manager Service controller is a graphical interface, which is exactly similar to the normal Manager application.

- Install the Manager via the Manager installer. Use the sa password noted previously to install the database during the installation.

Configuration & Operation

- The Manager can be run from the desktop or Start -> Programs -> Alchemi.NET -> Manager ->
- Alchemi.NET Manager. The database configuration settings used during installation automatically appear when the Manager is first started.
- Click the "Start" button to start the Manager.
- When closed, the Manager is minimized to the system tray.

2. Cross Platform Manager

Installation

- Install the XPManager web service via the Cross Platform Manager installer.

Configuration

- If the XPManager is installed on a different machine that the Manager, or if the default port of the Manager is changed, the web service's configuration must be modified. The XPManager is configured via he ASP.NET Web.config file located in the installation directory (wwwroot\Alchemi.NET\CrossPlatformManager by default):

```
<appSettings>
<add key="ManagerUri" value="tcp://localhost:9000/Alchemi.NET_Node" />
</appSettings>
```

Operation

- The XPManager web service URL is of the format http://[host_name]/[installation_path]
- The default is therefore

http://[host_name]/Alchemi.NET/CrossPlatformManager

- The web service interfaces with the Manager. The Manager must therefore be running and started for the web service to work.

3. Executor

Installation

The Alchemi.NET Executor can be installed in two modes

- As a normal Windows desktop application
- As a windows service. (Supported only on Windows NT/2000/XP/2003)

- To install the executor as a windows application, use the Executor Setup installer. For service mode installation uses the Executor Service Setup. The configuration steps are the same for both modes. In case of the service-mode, the "Alchemi.NET Executor Service" installed and configured to run automatically on Windows start-up. After installation, the standard Windows service control Manager can be used to control the service. Alternatively the Alchemi.NET ExecutorServiceController program can be used. The Executor service controller is a graphical interface, which looks very similar to the normal Executor application.
- Install the Executor via the Executor installer and follow the on-screen instructions.

Configuration & Operation

The Executor can be run from the desktop or Start -> Programs -> Alchemi.NET -> Executor -> Alchemi.NET Executor. The Executor is configured from the application itself. You need to configure 2 aspects of the Executor:

- The host and port of the Manager connect to Dedicated / non-dedicated execution.A non-dedicated Executor executes grid threads on a voluntary basis (it requests threads to execute from the Manager), while a dedicated Executor is always executing grid threads (it is directly provided grid threads to execute by the Manager). A non-dedicated executor works behind firewalls and
- Click the "Connect" button to connect the Executor to the Manager.

REFERENCES

[1] Rajkumar Buyya and Srikumar venugopal , "A Gentle Introduction to Grid Computing and Technologies",http://www.buyya.com/papers/GridIntro-CSI2005.pdf.

[2] C.Kesselman. I. Foster. Computational grids. In The Grid: Blueprint for a New Computing Infrastructure., chapter 2. Morgan-kaufman edition, 1999.

[3] J. Joseph, C. Fellenstein, "Grid Computing", Prentice Hall/IBM Press, Edition 2004

[4] Ferreir,L,Bieberstein,N.,berstis,V.Armstrong,J."Introjuction to grid computing with Globus",Redbook IBM corporation , http://www.www.redbooks.ibm.com.redbooks/pdfssg246895.pdf.

[5] Mark Baker, Rajkumar Buyya and Domenico Laforenza ,"The Grid: International Efforts in Global Computing".

[6] Manish Parashar, The Applied Software system Laboratory Rutgers," Grid Computing: My View" The State University of Jersey http://www.caip.rutgers.edu/TASSL/Papers/gc-overview-mp-09-03.pdf

[7] Duane Nickull "Service Oriented Architecture Whitepaper" Adobe Systems Incorporated, 2004.

[8] Jean-Christophe Durand Grid Computing 'A Conceptual and Practical Study' November 8, 2004

[9] Ian Foster "What is the Grid? A Three Point Checklist" Argonne National Laboratory & University of Chicago, http://www.gridbus.org/papers/TheGrid.pdf.

[10] David De Roure and others, "The Semantic Grid: Past, Present and Future", Proceedings of the 2nd Annual European Semantic Web Conference (ESWC2005), Volume 93, Issue 3, 2005.

[11] Arie Shoshani, Alex Sim and Junmin Gu, Lawrence Berkeley National aboratory, Storage Resource Managers: Middleware Components for Grid Storage.

[12] Hai Zhuge, Xiaoping Sun, Jie Liu, Erlin Yao, and Xue Chen , A Scalable P2P Platform for the Knowledge Grid . http://portal.acm.org/citation.cfm?id=1098620

[13] Scheduling and Resource Management in Computational Mini-Grids,July 1, 2002 Yih-Jiun Lee and Peter Henderson, DSSE research group,.Electronics and Computer Science ,University of Southampton , A Modelling Notation for Grid Computing.

[14] Inderpreet Chopra ,"Fault Tolerance in Computational Grids" TIET,Punjab.

[15] Thierry Prioi, "Grid Middleware", Advanced Grid Research Workshops through European and Asian Co-operation. http://www.gridatasia.net/download/Beijing/22/Workshop-Middleware-Priol.pdf

[16] Russell Lock, "An introduction to the Globus toolkit",http://www.comp.lancs.ac.uk/computing/research/cseg/projects/dirc/papers/An%20 ntroduction%20to%20the%20Globus%20toolkit.doc

[17] Foster and C. Kesselman, "The Globus Project: a Status Report", In Proc. of Seventh Heterogeneous Computing Workshop (HCW 98), IEEE Computer Society Press, March, 1998.

[18] Adam Jamison Oliner .Cooperative Checkpointing for Supercomputing Systems by Submitted to the Department of Electrical Engineering and Computer Science on May 19, 2005.

[19] Alan Wood, Swami Nathan, Timothy Tsai, Chris Vick, Lawrence Votta – Sun Microsystems;Anoop Vetteth – University of Illinois* Multi-Tier Checkpointing for Peta-Scale Systems

[20] Paul Townend and Jie Xu-"Fault Tolerance within a Grid Environment",University of Durham, DH1 3LE, United Kingdom

[21] Varun Sekhri, "CompuP2P: A light-weight architecture for Internet computing" MS Thesis, Iowa State University, Iowa, 2005

[22] Gengbin Zheng, Chao Huang and Laxmikant V. Kal´e .Department of Computer Science University of Illinois at UrbanaChampaign Performance Evaluation of Automatic Checkpointbased Fault Tolerance for AMPI and Charm++ .

[23] Thierry Prioi, "Grid Middleware", Advanced Grid Reseach Workshops through Europeon and Asian Co-operation. http://www.gridatasia.net/download/Beijing/22/Workshop-Middleware-Priol.pdf

[24] Klaus Krauter, Rajkumar Buyya and Muthucumaru Maheswaran,"A taxonomy and survey of grid resource management systems for distributed computing" Copyright 2001 John Wiley & Sons, Ltd. 17 September 2001.

[25] Kamana Sigdel, "Resource Allocation in Heterogeneous and Dynamic Networks" MS Thesis, Delft University of Technology,2005

[26] Sriram Krishnan, "An Architecture for Checkpointing and Migration of Distributed Components on the Grid" PhD Thesis, Department of Computer Science, Indiana University, November 2004

[27] J.H. Abawajy,"Fault-Tolerant Scheduling Policy for Grid Computing Systems", IPDPS'04 .

[28] Y. Tamir and C. Equin. Error recovery in multicomputers using global checkpoints. In 13rh Inremarional Conference on Parallel Processing, pages 32-41, August 1984.

[29] www.Alchemi.NET.net

[30] Y. M.Wang and W. K. Fuchs. Optimisitic message log-ging for independent checkpointing in message-passing systems. In Proceedings of the 11th Symposium on Reliable Distributed Systems, pages 147{154, October 1992.

[31] M. Chandy and L. Lamport. Distributed snapshots:determining global states of distributed systems. ACM Transactions on Computer Systems, 3(1):63{75, February 1985.

[32] www.globus.com

[33] http://www.cs.wisc.edu/condor/overview/

[34] Krishna Nadiminti, Akshay Luther, Rajkumar Buyya, "Alchemi.NET: A .NET- based Enterprise Grid System and Framework" December 2005.

[35] Douglas Thain, Todd Tannenbaum, and Miron Livny, "Distributed Computing in practice: The Condor Experience" Concurrency and Computation: Practice and Experience, Vol. 17, No. 2-4, pages 323-356, February-April, 2005.
http://www.cs.wisc.edu/condor/publications.html

[36] www.wiki.gridpp.ac.uk/wiki/Grid_middleware

[37] Francois Grey, Matti Heikkurinen, Rosy Mondardini, Robindra Prabhu, "Brief History of Grid",

http://Gridcafe.web.cern.ch/Gridcafe/Gridhistory/history.html.

[38] Rajkumar Buyya, "The Nimrod-G Resource Broker: an economic Based Grid Scheduler" http://www.buyya.com/thesis/Gridbroker.pdf

[39] Gengbin Zheng, Chao Huang and Laxmikant V. Kal´e." Performance evaluation of Automatic Checkpointbased Fault Tolerance for AMPI and Charm++".

[40] Markus Nilsson-A tool for automatic formal analysis of fault tolerance.

[41] http://www.globus.org/research/papers/anatomy.pdf

[42] Floyd Piedad, M.H., "High Availability: Design, Techniques and Processes". Enterprise computing series: Printice Hall PTR,2002

[43] Ya-Fing Zhung, Jizhou Sun, Jian-Bo Ma, "Self Management Model Based on multiagent and Worm Techniques" CCECE 2004

[44] Foster, I., Kesselman, C. and Tuecke, S., "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," International Journal of High Performance Computing Applications, 15 (3). 200-222. 2001.

[45] Francois Grey, Matti Heikkurinen, Rosy Mondardini, Robindra Prabhu, "BriefHistoryofGrid", http://Gridcafe.web.cern.ch/Gridcafe/Gridhistory/history.html

[46] http://www.gridcomputingplanet.com/

[47] Liang PENG, Lip Kian NG, "N1GE6 Checkpointing and Berkeley Lab Checkpoint/Restart" Dec 28, 2004

[48] Xianan Zhang, Dmitrii Zagorodnov, Matti Hiltunen, Keith Marzullo, Richard D. Schlichting, "An Agent Oriented Proactive Fault-tolerant Framework for Grid Computing", Universiry of Califonia,San Diego and AT&T Research Laboratory,2003.

[49] en.wikipedia.org/wiki/Grid_computing

[50] Algirdas Avizˇ ienis, Jean-Claude Laprie, Brian Randell- Fundamental Concepts of Dependability.

[51] Foster, I., Kesselman, C., "The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann", pp. 15-51, 1999

[52] http://wikis.sun.com/display/GridEngine/Using+Job+Checkpointing

[53] Paul Townend and Jie Xu, "Fault Tolerance within a Grid Environment"

[54] Chaitanya Kandagatla ,University of Texas, Austin. “Survey and Taxonomy of Grid Resource Management Systems”

[55] GRMS, http://www.gridworkflow.org/snips/gridworkflow/space/GRMS

[56] http://www.gridcomputingplanet.com/

[57] en.wikipedia.org/wiki/Grid_computing

[58] Mangesh Kasbekar Chandramouli Narayanan Chita R Das, “Selective Checkpointing and Rollbacks in Multithreaded Object-oriented Environment” 6th IEEE International On-Line Testing Workshop (IOLTW) p. 3

Paper Published

Neeraj Kumar Rathore, Ms. Inderveer Chana, "**COMPARATIVE ANALYSIS OF CHECKPOINTING**" PIMR Third National IT Conference - 2008 on "Prestige Institute of Management and Research ", Indore (M.P.), India. (Accepted).