**“FAULT TOLERANCE IN DISTRIBUTED SYSTEMS”**

*Thesis submitted to*

*Visvesvaraya National Institute of Technology, Nagpur*

*In partial fulfillment of requirement for the award of*

*degree of COMPUTER SCIENCE AND ENGINEERING*

**Bachelor of Technology**

**(Department of Computer Science & Engineering )**

*by*

**Ankita Raman (BT06CSE011)**

**Kamlesh Yadav (BT08CSE035)**

**Nagalla Sruthi (BT08CSE044)**

**Sanjay Singh (BT08CSE060)**

*Guide*

**MANISH KURHEKAR**



**Department of Electronics and Computer Science Engineering**

**Visvesvaraya National Institute of Technology**

**Nagpur 440 010 (India)**

**May 2012**

**DECLARATION**

I, hereby declare that the thesis titled “**Fault Tolerance in Distributed Systems”** submitted herein has been carried out by me in the Department of Electronics and Computer Science Engineering of Visvesvaraya National Institute of Technology, Nagpur. The work is original and has not been submitted earlier as a whole or in part for the award of any degree / diploma at this or any other Institution / University.

**Ankita Raman (BT08CSE010)**

**Kamlesh Yadav (BT08CSE035)**

**Sruthi Nagalla (BT08CSE044)**

**Sanjay Singh (BT08CSE060)**

**CERTIFICATE**

The thesis titled “**Fault Tolerance in Distributed Systems”** submitted by

**Ankita Raman (BT08CSE010)**

**Kamlesh Yadav (BT08CSE035)**

**Sruthi Nagalla (BT08CSE044)**

**Sanjay Singh (BT08CSE060)**

for the award of degree of Bachelor of Technology has been carried out under my supervision at the Department of Computer Science Engineering of Visvesvaraya National Institute of Technology, Nagpur. The work is comprehensive, complete and fit for evaluation.

**Manish Kurhekar**

Associate Professor,

Department of Computer Science & Engineering,

VNIT, Nagpur

**Head**

Department of Electronics and Computer Science Engineering

VNIT, Nagpur

*Date: April 2012*

**Acknowledgment**

I express my heartfelt gratitude to Professor In charge Mr. Manish Kurhekar for giving me the opportunity to bring this project to realization and offering invaluable help in guiding us through. I am also thankful to all the members of the DSPD lab for providing me with a great atmosphere in the lab that made the daily work on this project a real pleasure.

A special thought also goes to all the Professors who have enlightened me with invaluable knowledge during my 3 years of stay in the department of Computer Science and Engineering.

**ABSTRACT**

Distributed system and occurrences of faults both are ubiquitous. As the size of distributed system is increasing day by day chances of faults are increasing . In large and dynamic distributed system millions of computing devices are working altogether and these millions of computing device are prone to failures. Faults are inevitable in larger and dynamic distributed system. Faults may stop or halt execution of distributed system. A fault renders a system to work abnormally, inefficiently leading to undesirable results or as in some cases might lead to the complete crash of the system defeating the entire purpose of the system. We cannot ever have a system which will be completely free from faults and failures. As a system can never be done away without occurrences of faults and failures it should be made tolerant .

A fault tolerant distributed computing aims at making distributed systems more reliable by catering for occurrence of faults which are undesirable but inevitable. The most egregious type of fault that will totally hinder the execution of a system is the system failure in the event of power failure or so. This is an effort towards making a distributed system resilient to such type of faults by designing a distributed client server model and making it fault tolerant.

List of Figures

*Figure 2.1 Basic Client Server Model*

*Figure 3.1 Client Sending request to server via socket*

*Figure 3.2 Server replying to Client request via socket*

*Figure 3.3 Socket Communication*

*Figure 3.4 : Final network after setting up databases in master-slave configuration*

*Figure 4.1 Network implementation*

*Figure 4.2 Control Transfer in case of Fault*

*Figure 5.1 Packet Structure*

*Figure 5.2 Message Flow*

*Figure 6.1 Possible Points of Failure*

Table of Contents

Chapter 1: Introduction

1.1. Distributed Systems

1.2 Fault tolerance

1.2.1 Motivation for Fault Tolerance system

1.2.2 Failures,Errors and Faults

1.2.3 Fault Classification

1.2.4 Significance in Distributed systems

1.2.5 Achieving Fault tolerance

Chapter 2: Client Server model in Distributed Systems

2.1 Introduction

2.2 Shortcomings

2.3 Improvisation

2.4 A Further improvement

Chapter 3: Underlying Concepts

3.1 Sockets

3.2 Database server

3.3 Connection of terminal server and database

Chapter 4:Database Concepts

4.1: Database Implementation

4.2: Connecting Terminal Servers with Database

Chapter 5: Implementation

5.1 Network Implementation

5.2 Design implementation

Chapter 6: Behaviour of fault tolerance approach-Connection transfer and message flow

6.1 Connection establishment

6.2 Forwarding data

6.3 Processing requests

6.4 Computation at the terminal side and updation at database

6.5 Message flow

Chapter 7: Analysis of Implementation

7.1 Possible points of failure

7.2 Fault tolerance

7.2.1 Client failure

7.2.2 Terminal server Failure

7.2.3 Database Failure

Chapter 8: Testing and Results of Fault Tolerant Approach

Chapter 9: Scope and Improvement areas

Chapter 10: Conclusion and References

**Chapter 1**

**INTRODUCTION**

As our high-tech society becomes increasingly dependent on computers, the demand for more dependable software will increase and likely become the norm. In the past, fault-tolerant computing was the exclusive domain of very specialized organizations such as telecom companies and financial institutions. With business-to-business transactions taking place over the Internet, however, we are interested not only in making sure that things work as intended, but also, when the inevitable failures do occur, that the damage is minimal.Systems used in critical applications such as health, commerce, transportation, utilities, and national security must be highly reliable. Ubiquitous use of computing systems and other electronic systems in these critical areas requires that computing systems have high reliability. High reliability is achieved by designing the systems to be fault-tolerant.

**1.1 Distributed Systems**

A distributed system consists of a collection of autonomous computers, connected through a network and distribution middleware, which enables computers to coordinate their activities and to share the resources of the system, so that users perceive the system as a single, integrated computing facility.

**Characteristics**

Following are some important characteristics of Distributed Systems :

* 1. Multiple autonomous components
  2. Components are not shared by all users
  3. Resources may not be accessible
  4. Software runs in concurrent processes on different processors
  5. Multiple Points of control
  6. Multiple Points of failure

**Importance**

Distributed systems supersedes others owing to following factors:

* Performance: A collection of processors can provide higher performance (and better Price performance ratio) than a centralized computer.
* Distribution: Many applications involve, by their nature, spatially separated machines (banking, commercial, automotive system).
* Reliability (fault tolerance): If some of the machines crash, the system can survive.
* Incremental growth: As requirements on processing power grow, new machines can be added incrementally.
* Sharing of data/resources: Shared data is essential to many applications (banking, computer supported cooperative work, reservation systems); other resources can be also shared (e.g. expensive printers).
* Communication: Facilitates human-to-human communication

**1.2 Fault Tolerance**

Fault tolerance is the ability of a system to perform its function correctly even in the presence of internal faults. The purpose of fault tolerance is to increase the dependability of a system. In practice we can never guarantee the flawless execution of tasks under any circumstances**.** Although system can undergo reduction in throughput or an increase in response time in the event of some partial failure. That is, the system as a whole is not stopped due to problems either in the hardware or the software.Very large number of failures means that the system will not produce useful results unless some fault-tolerance is incorporated.

Providing fault-tolerant design for every component is normally not an option. In such cases the following criteria may be used to determine which components should be fault-tolerant:

*How critical is the component?*

*How likely is the component to fail?*

*How expensive is it to make the component fault-tolerant?*

**1.2.1 Motivation for Fault Tolerance**

In the traditional approach (fault intolerance) the reliability of computing is assured by a priori elimination of the causes of unreliability, i.e. of faults. This elimination takes place before regular use begins, and the resources that are allocated to attain reliability are spent on perfecting the system prior to its field use. Redundancy is not employed, and all parts of the system must function correctly at all times. Since in practice it has not been possible to assure the complete a priori elimination of all causes of unreliability, hence the goal of fault intolerance is not fulfilled.

An alternate approach which alleviates most of the above shortcomings of the traditional fault-intolerance approach is offered by fault tolerance. In this approach, the reliability of computing is assured by the use of protective redundancy. Reliable computing is made possible despite certain classes of hardware failures, external interference with computer operation, and perhaps even remaining design errors in hardware and software. So the main reason to the use of a fault-tolerant design is to achieve a reliability or availability prediction that cannot be attained by the purely fault-intolerant design.

**1.2.2 Failures, Errors and Faults**

In everyday language, the terms fault, failure, and errorare used interchangeably. In fault-tolerant computing system, however, they have distinctive meanings.

A *failure* occurs when an actual running system deviates from this specified behavior. The cause of a failure is called an *error*. An error represents an invalid system state, one that is not allowed by the system behavior specification.

The error itself is the result of a defect in the system or fault. In other words, a *fault* is the root cause of a failure. That means that an error is merely the symptom of a fault. A fault may not necessarily result in an error, but the same fault may result in multiple errors. Similarly, a single error may lead to multiple failures.

Thus the manifestation of failures, faults and errors follows a “fundamental chain”:-

*failure---> fault----> error-----> failure--->fault---->*

**System attributes**

System attributes which are to be considered in fault tolerance system are as following :

* Availability – system always ready for use, or probability that system is ready or available at a given time.
* Reliability – property that a system can run without failure, for a given time
* Safety – indicates the safety issues in the case the system fails
* Maintainability – refers to the ease of repair to a failed system

**Failure in a distributed system** occurs when a service cannot be fully provided .System failure may be partial or a single failure may affect other parts of a system (failure escalation)

**1.2.4 Significance of fault tolerance distributed Systems**

From a fault-tolerance perspective, distributed systems have a major advantage: They can easily be made redundant, which is at the core of all fault-tolerance techniques. Unfortunately, distribution also means that the imperfect and fault-prone physical world cannot be ignored, so that as much as they help in supporting fault-tolerance, distributed systems may also be the source of many failures.These can be

**1.2.5 Achieving fault tolerance**

The general approach to building fault tolerant systems is redundancy. Redundancy may be applied at several levels.

**Information redundancy** seeks to provide fault tolerance through replicating or coding the data. For example, a Hamming code can provide extra bits in data to recover a certain ratio of failed bits.

**Time redundancy** achieves fault tolerance by performing an operation several times. Timeouts and retransmissions in reliable point-to-point and group communication are examples of time redundancy. This form of redundancy is useful in the presence of transient or intermittent faults. It is of no use with permanent faults. An example is TCP/IP’s retransmission of packets.

Physical redundancy deals with devices, not data. We add extra equipment to enable the system to tolerate the loss of some failed components. RAID disks and backup name servers are examples of physical redundancy.

When addressing **physical redundancy**, we can differentiate redundancy from replication. With replication, we have several units operating concurrently and then a voting system to select the outcome. With redundancy, one unit is functioning while others are available to fill in in case the unit ceases to work.

**Chapter 2**

**THE CLIENT SERVER MODEL**

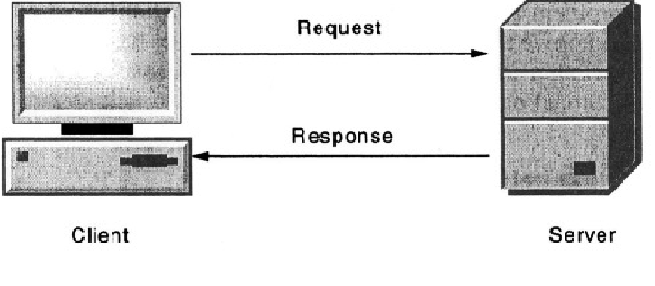
**2.1 Introduction**

A typical client-server model in Distributed Systems constitutes of :

***Service****:* A service is a software entity that runs on one or more machines providing an abstraction of a set of well-defined operations in response to applications’ requests.

***Server****:* A server is an instance of a particular service running on a single machine.

***Client****:* A client is a software entity, running on entirely different machine from that of the server, that exploits services provided by servers. A client can but does not have to interface directly with a human user.



*Figure 2.1 Basic Client Server Model*

According to this model there are two processes, the client, which requests a service from another process, and the server, which is the service provider. The server performs the requested service and sends back a response. This response could be a processing result, a confirmation of completion of the requested operation or even a notice about a failure of an operation.

**Features and Problems of the Client-Server Model**

The client-server model is characterized by its simplicity, modularity, extensibility and flexibility. The whole distributed computing system developed based on the client-server model can be easily extended by adding new services in the form of new servers. The servers which do not satisfy user requirements can be easily modified or even removed.

However it also faces the following shortcomings and challenges

**2.2 Shortcomings**

* The control of all the individual resources in a system is centralized on a single server. This means that if the computer supporting the server fails, that element of control fails. Such a solution is not tolerable if a control function of a server is critical to the operation of the system (e.g., a name server, a file server, an authentication server).
* The second problem is that a single server is a potential bottleneck. The problem is exacerbated as more computers with potential clients are added to the system.

**2.3 Improvisation**

**Improvements over the client server model**

To meet the above shortcomings the client server model is modified and ameliorated suitably as :

1. Addition of more servers so that the service provided to a client is not restricted to just one server. The additional servers are equally capable of providing the same desired service .

This further helps in distributing the load of incoming clients over the computational servers. Thus the high probability of crashing of a server due to overload is averted.

This is facilitated by having a main server in between the clients and computational servers which distributes the load of the client requests over the computational servers, henceforth referred to as terminal servers.

**2.4 A further improvement**

To ensure computation for a client to be carried out independent of other clients irrespective of the fact to which terminal server its requests get forwarded to, its session and states must be stored which is achieved by

2. a set of synchronized databases connected to the servers, to maintain the session and state information corresponding to each client requesting service and in connection.

**Chapter 3**

**UNDERLYING CONCEPTS**

Moving on from the design implementation we have the software and hardware implementation which makes our design model a working model.

Software implementation is the realization of technical specification or algorithm as a program.

The basic requirement for a distributed system to work is the reliable communication between the various processes which may be residing on different machines.To be able to send the messages across the network of participating processes or the machines on which they run , we bring into context the concept of socket programming.

**SOCKET**

Socket is the software endpoint that establishes bidirectional communication between the systems in a distributed network constituting distributed system.

It is a software abstraction of a port that serves as an endpoint of interprocess communication.

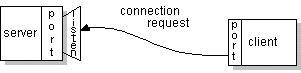
A socket associates the server program with a specific hardware port on the machine where it runs so any client program anywhere in the network with a socket associated with that same port can communicate with the server program.

There are several Internet socket types available:

* Datagram sockets, also known as connectionless sockets, which use User Datagram Protocol (UDP)
* Stream sockets, also known as connection-oriented sockets, which use Transmission Control Protocol (TCP) or Stream Control Transmission Protocol (SCTP).
* Raw sockets (or Raw IP sockets), typically available in routers and other network equipment. Here the transport layer is bypassed, and the packet headers are made accessible to the application.

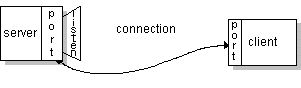
Normally, a server runs on a specific computer and has a socket that is bound to a specific port number. The server just waits, listening to the socket for a client to make a connection request.

*On the client-side:* The client knows the hostname of the machine on which the server is running and the port number on which the server is listening. To make a connection request, the client tries to rendezvous with the server on the server's machine and port.It initiates a TCP connection to the server by creating a socket object(three-way handshake). The client also needs to identify itself to the server so it binds to a local port number that it will use during this connection. This is usually assigned by the system.



*Figure 3.1 Client Sending request to server via socket*

Server welcomes some initial contact from a client using Welcoming socket and Connection socket is created at initial contact of client. If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket bound to the same local port and also has its remote endpoint set to the address and port of the client. It needs a new socket so that it can continue to listen to the original socket for connection requests while tending to the needs of the connected client.



*Figure 3.2 Server replying to Client request via socket*

On the client side, if the connection is accepted, a socket is successfully created and the client can use the socket to communicate with the server.

The client and server can now communicate by writing to or reading from their sockets.

An endpoint is a combination of an IP address and a port number. Every TCP connection can be uniquely identified by its two endpoints. That way we can have multiple connections between our host and the server.

The java.net package in the Java platform provides a class, Socket, that implements one side of a two-way connection between our Java program and another program on the network. The Socket class sits on top of a platform-dependent implementation, hiding the details of any particular system from our Java program. By using the java.net.Socket class instead of relying on native code, our Java programs can communicate over the network in a platform-independent fashion.

It has following constructor and methods

* Socket(String host, int port): Creates a stream socket and connects it to the specified port number on the named host.
* InputStream getInputStream()
* OutputStream getOutputStream()
* close()

Additionally, java.net includes the ServerSocket class, which implements a socket that servers can use to listen for and accept connections to clients. It waits for requests to come in over the network.Then performs some operation based on the request.

Constructor and methods available in it are as following

* ServerSocket(int port)
* Socket Accept(): Listens for a connection to be made to this socket and accepts it. This method blocks until a connection is made.

### Programming sockets in Java

For programming a client, open a socket like this:

Socket MyClient;  
MyClient = new Socket("Machine name", PortNumber);

Where *Machine name* is the machine we are trying to open a connection to, and PortNumber is the port (a number) on which the server we are trying to connect to is running. When selecting a port number, it should noted that port numbers between 0 and 1,023 are reserved for privileged users (that is, super user or root). These port numbers are reserved for standard services, such as email, FTP, and HTTP. When selecting a port number for our server,we select one that is greater than 1,023.

For programming a server, this is how we open a socket:

ServerSocket MyService;

MyServerice = new ServerSocket(PortNumber);

When implementing a server we also need to create a socket object from the ServerSocket in order to listen for and accept connections from clients.

Socket connectionSocket = null;

connectionSocket = MyService.accept();

*Creating input/output streams so that client and server can receive/send data to each other:*

*INPUT :*

On the client side, we can use the DataInputStream class to create an input stream to receive response from the server:

DataInputStream input = new DataInputStream(MyClient.getInputStream());

The class DataInputStream allows us to read lines of text and Java primitive data types in a portable way. It has methods such as read, readChar, readInt, readDouble, and readLine. We can use any function depending on the type of data received from the server.

On the server side, we can use DataInputStream to receive input from the client:

DataInputStream input = new DataInputStream (connectionSocket.getInputStream());

*OUTPUT:*

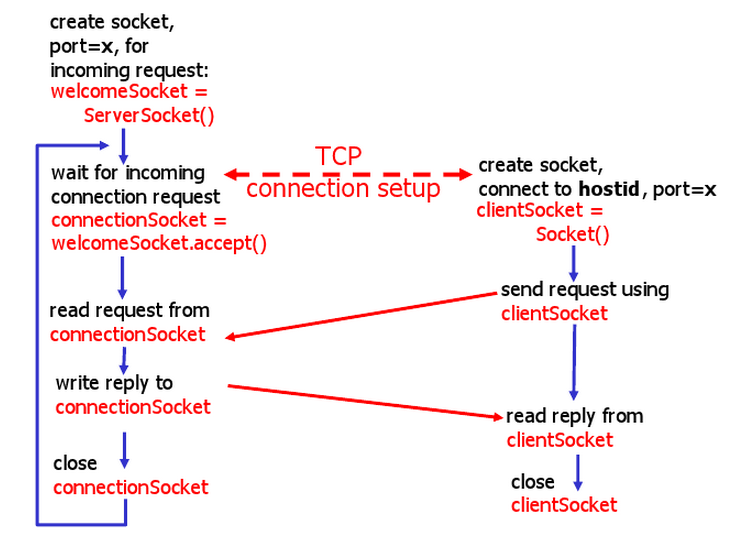
On the client side, we can create an output stream to send information to the server socket using the class PrintStream orDataOutputStream of java.io:

DataOutputStream output = new DataOutputStream(MyClient.getOutputStream());

The class *DataOutputStream* allows us to write Java primitive data types; many of its methods write a single Java primitive type to the output stream. The method writeBytes is a useful one.

On the server side, we can use the class DataOutputStream as mentioned above.

Output and input stream should always be closed before the socket is closed.

**Server Side Client Side**

*Figure 3.3 Socket Communication*

**Chapter 4**

**DATABASE CONCEPTS**

**Database implementation**

The servers are connected with a set of synchronized databases for maintaining and storing state of each client in connection with them.

We have used three MySQL Database 5.5 for our implementation.

**Installing MySQL**

On Debian and related distributions, there are two packages, **mysql-client** and **mysql-server**, for the client and server components respectively.To download and install, including any dependencies, use the apt-get command, specifying the packages that are to be installed.

A sample installation of the MySQL packages might look like this :

**root-shell> apt-get install mysql-client-5.5 mysql-server-5.5**  
Reading package lists... Done  
Building dependency tree   
Reading state information... Done  
The following extra packages will be installed:  
 bsd-mailx ........ mysql-common postfix  
Suggested packages:  
 dbishell ............. resolvconf postfix-cdb  
The following NEW packages will be installed  
 bsd-mailx .............. mysql-server-5.5 postfix  
0 upgraded, 13 newly installed, 0 to remove and 182 not upgraded.  
.

.

\* Starting MySQL database server mysqld ...done.  
\* Checking for corrupt, not cleanly closed and upgrade needing tables.  
Processing triggers for libc6 ...  
ldconfig deferred processing now taking place

During installation, the initial database will be created, and user will be prompted for the MySQL root password (and confirmation). A configuration file will be created in /etc/mysql/my.cnf. An init script will have been created in /etc/init.d/mysql.

The server can be started and stopped using following command :

root-shell> service mysql [start|stop]

To avoid data loss in case of database failure we synchronise all databases. As replication provides flexible topologies for scale-out and high availability and enables users to cost-effectively deliver application performance , we use master slave replication for database synchronisation.It enables data from one MySQL server, called the master, to be replicated to one or more MySQL servers, called slaves.This is not a backup policy because an accidentally issued DELETE command will also be carried out on the slave.

Broadly speaking replication is basically a three step process which works as follows:

* The master records changes to data in its binary log.
* The slave copies the changes recorded in the master’s binlog to its relay log.
* Then the slave replays the change-set recorded in its relay log, applying these changes to its own data.

We implement master slave replication in such a way that every database is master of all the other databases in network so that change in any database is reflected on others also.

The replication involves following steps :

### Configuring the master

First we have to edit */etc/mysql/my.cnf*. We have to enable networking for MySQL, and MySQL should listen on all IP addresses, therefore we comment out these lines (if existent):

#skip-networking

#bind-address = 127.0.0.1

The next step is to enable binary logging, specify the server id and log file that should be used. These logs are used by the slave to see what has changed on the master

log-bin = /var/log/mysql/mysql-bin.log

binlog-do-db = <database name>

server-id = 1

*log-bin* is the binary log basename to generate binary log file names.

*binlog-do-db* is the database that is to be replicated on slave side.

*server-id* option is used in replication to enable master and slave servers to identify themselves uniquely.

Then we restart MySQL:

/etc/init.d/mysql restart

### Setting up replication accounts

The next step to do is to create a user account on the master and give it the proper privileges, so that the slave I/O thread can connect to the master and read master’s binary log.

mysql> GRANT REPLICATION SLAVE, REPLICATION CLIENT ON \*.\* TO replicator@'<slave IP>' IDENTIFIED BY '<password>';

The REPLICATION SLAVE privilege should be given to accounts that are used by slave servers to connect to the master server. Without this privilege, the slave cannot request updates that have been made to databases on the master server.

The REPLICATION CLIENT privilege enables the use of SHOW MASTER STATUS and SHOW SLAVE STATUS.

**Configuring the Slave**

After setting up the master the next step is to setup slave so that its ready for replication.

The slave requires the following configuration:

server-id = 2

master-host = <master IP>

master-user = <MySQL user>

master-password = <password>

master-connect-retry = 60

replicate-do-db = <database name>

*master-host* is the IP address or hostname of the master

*master-user* is the user we granted replication privileges on the master.

*master-password* is the password of *master-user* on the master.

Then we restart MySQL:

/etc/init.d/mysql restart

Now to import existing master database on slave side do

LOAD DATA FROM MASTER;

And this completes configuring slave. In order to change master of a slave server use following command after importing database from master server:

CHANGE MASTER TO MASTER\_HOST='<new-master-IP>', MASTER\_USER='<MySQL-user>' MASTER\_PASSWORD='<password>', MASTER\_LOG\_FILE='<log-file-name.log>';

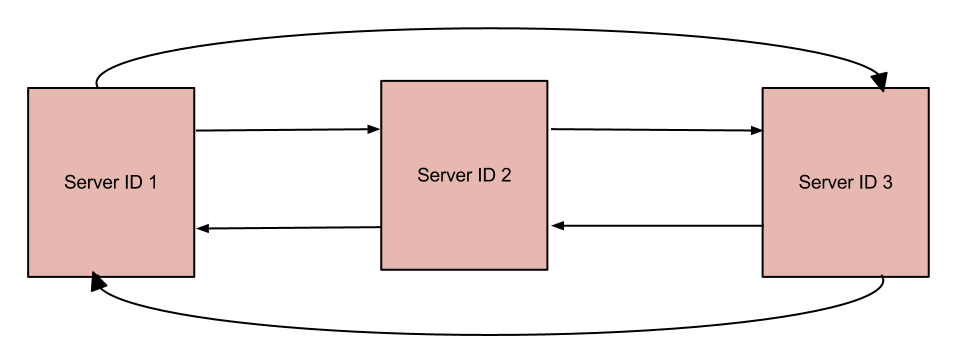
*master-user* is the user we granted replication privileges on the master.

*master-log-file* is the file MySQL gave back when we ran *SHOW MASTER STATUS;* on the   
master.

*master-log-pos* is the position MySQL gave back when we ran *SHOW MASTER STATUS;* on the master.

Similarly we configure the other database also to be the slave of the same master server with server id 3 so that we have two slaves for a given master.

In case the master server fails, all other databases are rendered useless.To deal with this situation we repeat the complete process to set as the other database as master and rest two as slave. This way we finally have a network of databases such that every database is master of all and at the same time is also the slave of all other databases in network. Now even if any one of the database fails rest all are still synchronised.



*Figure 4.1 : Final network after setting up databases in master-slave configuration*

**CONNECTING TERMINAL SERVERS WITH DATABASE**

After setting up databases in master slave configuration next step is to connect terminal server to database servers so that it can fetch and store content from database.

MySQL provides connectivity for client applications developed in the Java programming language through a JDBC driver, which is called MySQL Connector/J.

#### Installing the Driver and Configuring the CLASSPATH

After downloading and extracting the distribution archive, install the driver by placing mysql-connector-java-<*version>*-bin.jar in classpath, either by adding the full path to it to **CLASSPATH** environment variable, or by directly specifying it with the command line switch **-cp** when starting the JVM.

root-shell> export CLASSPATH=/path/mysql-connector-java-***ver***-bin.jar:$CLASSPATH

For using the driver with the JDBC **DriverManager**, use **com.mysql.jdbc.Driver** as the class that implements **java.sql.Driver**.

**Chapter 5**

**IMPLEMENTATION**

We aim at designing a distributed system which provides a simple service such as facilitating a client to perform arithmetic operations on the data(numbers) sent by it.

**Possible Shortcomings and Challenges in our Design**

* Overloading of Server :- If there are multiple client requests at the same time and server has not enough resources to handle those requests, server failure may take place due to overloading of number of requests.
* Distinguishing Between different clients:- We are using multithreaded model in which multiple clients can request at same time, so we need some criteria to distinguish these client requests and maintain unique state for each client.

Also the system is to be made fault tolerant such that the client is oblivious to server side failures and in the event of failure client is kept transparent to all the operations and connection transfers and the latency is to be kept low

**5.1 Network Implementation**

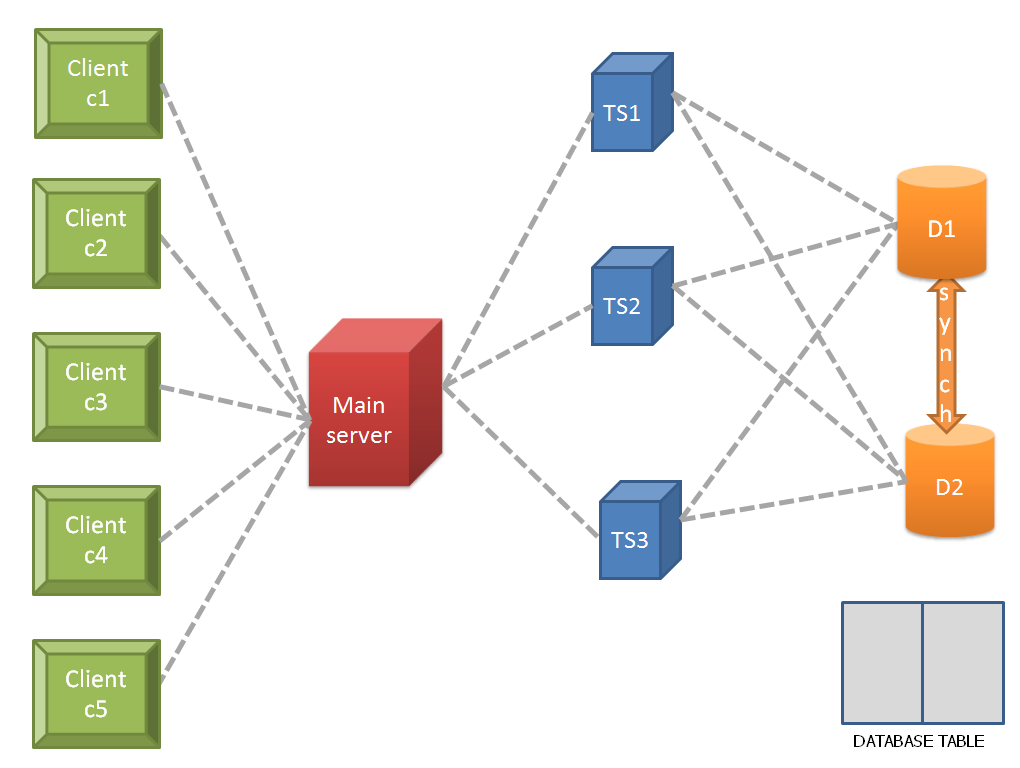
For our implementation we chose 2 tier client server architecture with five clients, one main server, three terminal servers and two database servers.

Each client is a TCP client which connects to main server via sockets.

The main server is connected to three different terminal servers set to perform same task. Main server and terminal server also communicates using sockets. Socket connections are established as described in underlying concepts.

Terminal servers are connected to database for providing stable storage media. This connection is made by java JDBC Connector.

The databases are in turn connected to each other and are synchronised by master slave method such that each server is master and at the same time is also the slave of the other database server.



*Figure 5.1 Network implementation*

**5.2 Design Implementation**

For the purpose of implementing a functional distributed system and then making it fault tolerant, we implemented a simple service of adding numbers on our system.

In it client sends a request to main server which consists of an identifier and a number. The main server redirects the request to terminal server where the number is added to previous sum of numbers from that client.

Then in order to make the current distributed system fault tolerant we use connected databases to store state information of each connection so as to deal with faults.The design and implementation is explained in detail below.

**Terms used in Model**

* **Service**- Service requested by a client is to perform mathematical operations on the data sent by the client.
* **Data**- Data sent by the client is an integer on which operation is to be done, sent over network one digit at a time.
* **State**- Result of any operation performed on the data sent by the client constitutes its state at that instant of time.
* **Session**- An entire set of transactions of a client while remaining in the connectivity with the same server constitutes a session.

Accordingly

* **State management-** After each operation the value of result is updated in the database hence state is the value corresponding to a client in connection stored in the database
* **Session management-** The result of the transactions carried by a client is stored and updated in the database as long as the client remains in connection and is cleared only after the client withdraws.

Our design for fault tolerance model consists of two main parts

* Design implementation for managing load and terminal server crash.
* Design implementation for managing states of clients and distinguishing between clients.

**Managing load and server crash**

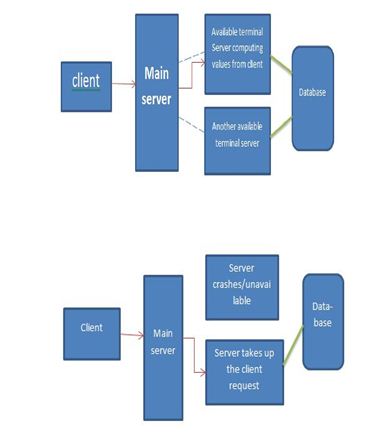
The clients’ requests are forwarded to the main server . The main server which is connected to numerous terminal servers, distribute these requests in a round robin fashion thus acting as a load balancer. The terminal servers are designed to provide the same functionality and service to a client in connection with it.

**Managing states of clients and distinguishing between clients**

A client’s request might not be forwarded to the same terminal server always .So care should be taken that current state value which is the result of the computations carried out for a particular client so far is sent to another terminal server which happens to receive a new service request from this client

This is facilitated by storing states of clients which is nothing but the value of result computed so far in the database along with the identifier. So with each new request coming from the client the server first looks up in the the database for the client’s state value and fetches the value for further computations to be performed , if an entry for the client in question exists

else it creates a new entry and set the current state value of the client to zero.



*Figure 5.2 Control Transfer in case of Fault*

**Chapter 6**

**Behaviour of Fault Tolerance Approach**

**Connection transfer**

The connection transfer and the flow of messages across the distributed system give a closer look and better understanding of how the overall system works.

**6.1 Connection establishment**

1) A client establishes a plenary session where in it requests service which is arithmetic operations to be performed over the data sent by the client. Assuming the servers to be always available connection establishment is done by directly sending the data to the server for computation.

If the client wants start a new transaction afresh it clears the result of previous transactions by

command - START.

**6.2 Forwarding data**

The request sent by the client is the data value to the server for computation which is encapsulated in a packet along with its ip address which acts an identifier. To achieve this the integer value is concatenated with the ip address as a string. and the entire string is sent as a packet.



*Figure 6.1 Packet Structure*

**6.3 Processing requests**

All the requests are directed to Main Server.No computation is performed at the main server, still the main server performs the following functions..

a) It keeps polling terminal servers to determine their availability for which it continuously

pings them after a predetermined time interval, 1 sec in our case. For the details,

an array holding all the ip addresses of the terminal server systems is maintained.

b) Any incoming request from a client i.e the data packet is simply forwarded to the very first available server

All other subsequent requests are forwarded to available servers in a round robin order..

c) The multithreaded main server creates new threads for each incoming clients so that concurrent processing is achieved while the main server keeps polling for the availability of the terminal servers.

If a terminal server already in connection with a client is found to unavailable during polling by main server it is marked as unavailable by setting the corresponding boolean flag to false and the thread processing it is informed about its unavailability.

**6.4 Computation at the terminal side and updation at database**

Terminal servers are responsible for carrying out computations on the data forwarded to it by the main server sent by the client. Although each terminal servers work independently of each other their functionalities is same :

a)the server takes the incoming packet and divides it into the required value and IP address

b)it then checks the database if attributes to this IP occurs .

c)if the IP doesn’t exist it adds a new row to the database corresponding to this IP and updates the other attribute fields.(like setting sum to 0).

d)If the IP already exists ,then the server fetches the stored sum attribute and does the required arithmetic computation (say addition ) with the value obtained from the request.

e)Prior to sending the result back to main server, it updates the sum field to the new computed value on the database.

f)The server operates all along until it is interrupted by a fault or the client terminates .

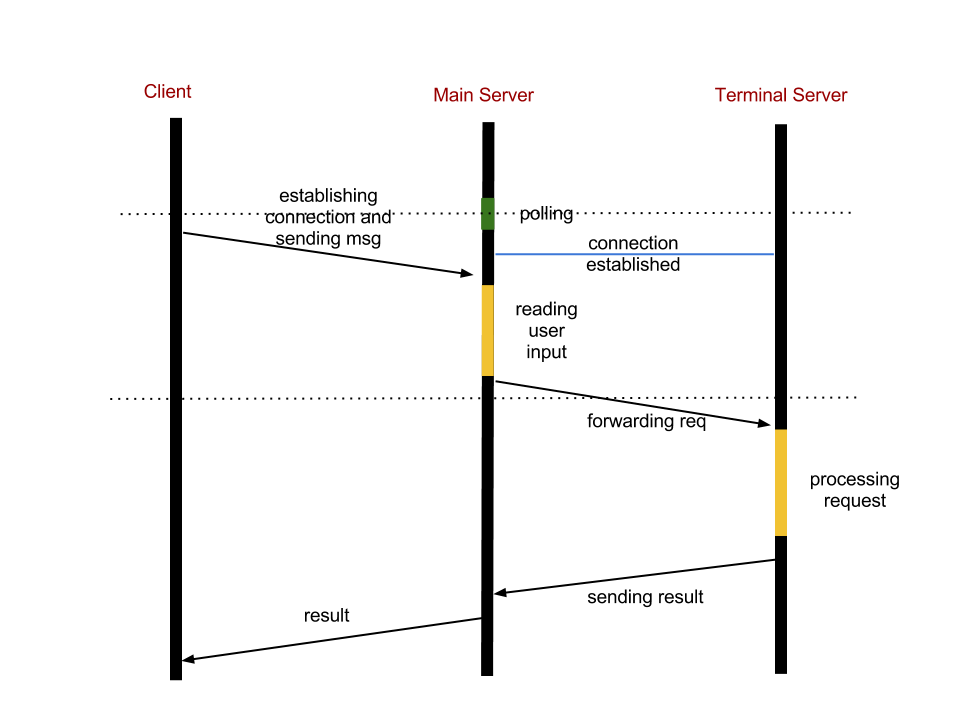
g)A single server is designed to handle multiple client connections.

**Response**

7)the results of the computation so calculated are transferred back to the client via the main server.

8)Hence the requests are catered in this fashion .

**6.5 Message flow**

The client , main server and terminal server communicate with each other as illustrated by following diagram 

*Figure 6.2 Message Flow*

Main server polls all terminal servers in network and wait on available server for incoming client. As the client sends requests to main server for processing data it accepts connection, reads input from client and then redirects the same data to terminal server on which it was waiting.The function of main server is limited to accepting requests from clients and redirecting them to available terminal servers. No processing/operation on data is done at main server side. Terminal server on receiving the forwarded request processes the request i.e. add number to the previous sum in our case. After processing the request and updating the state information the terminal server replies to main server from where the reply containing the result(sum in our case) is forwarded to the client.This is how message flows when we have single threaded main server which caters one request at a time. In order to make main server to listen to multiple clients we make main server multithreaded. So as soon as the main server gets request from client it threads. Now the newly created thread is responsible for handling main server’s task of sending the reply back to client. Meanwhile the main thread starts polling for available terminal server so that it can accept request from another client.

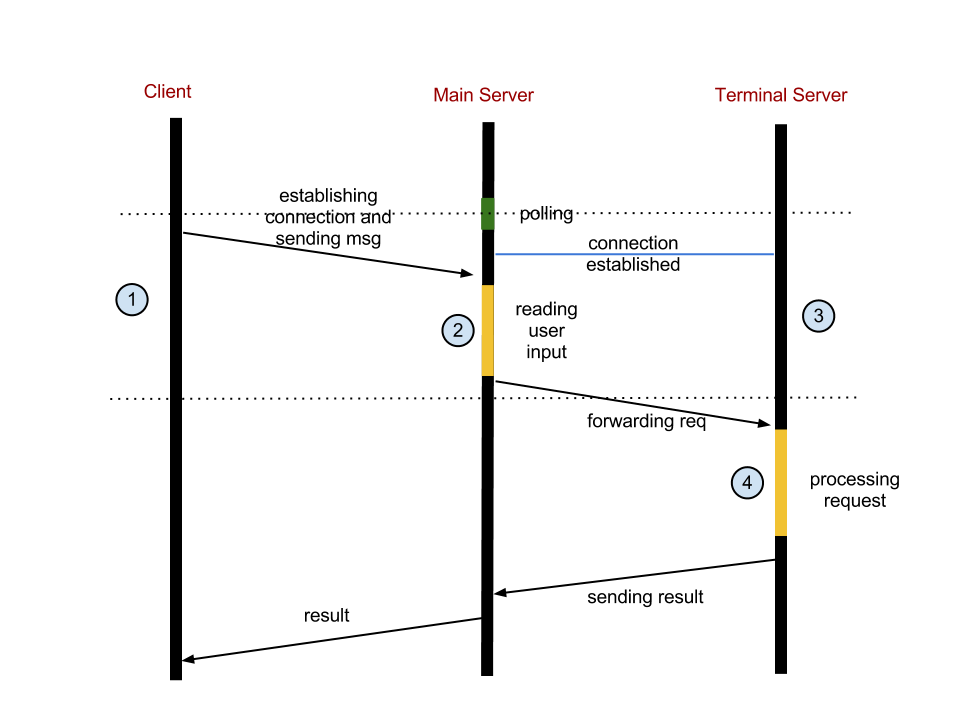
**Chapter 7**

**ANALYSIS OF IMPLEMENTATION**

**7.1 Possible points of failure**

As discussed earlier multiple servers and synchronized databases were implemented in order to deal with faults of servers and databases. Now we analyse our implementation by considering all possible points of failure and performance under those conditions.

Following figure shows possible failure points in our implementation :



*Figure 7.1 Possible Points of Failure*

**Client failure**

* A client can fail after sending request and before receiving reply from server as shown in fig by point 1.
* A client can even fail after accepting reply or before sending request.

**Terminal server failure**

* Terminal server may fail after accepting connection from a main server and before clients’ request gets forwarded to it (3)or
* amidst processing clients’ request.(4)

**Database Failure**

* A database may fail at any instant of time

**7.2 Fault Tolerance**

As we have designed our network to be fault tolerant, all the above mentioned failures are dealt properly as mentioned below:

**Client failure**

As client can fail in two different situations, considering each point of failure one by one. If client fails at point (1) then during recovery client need not do anything. It can start afresh and get the result of previous processing just by sending number 0. As the states are already stored in database it can continue from the point it crashed.

If client fails before sending request or after receiving reply, in that case since client was idle it can just make a fresh start as in previous case.

**Terminal server failure**

As stated earlier terminal server can fail at two different places.

If it fails at point(3) then thread polls for some other available terminal server and establishes connection with the first available server. All requests are then forwarded to that server.Since states are stored on stable storage i.e. database the new server assigned can access the previous state information related to that client and continue from the place where the other server crashed. Hence even if the terminal server crashes client’s request is still processed without any loss of data and client is unaware of this control transfer.

If it fails at point(4) then the main server won’t get any reply, so after a specified amount of time the main server will time out and will start polling for other available server. It then forwards the request (client’s data which was stored in its buffer) to the available server and get the request processed.

**Database failure**

Database failure was already dealt by synchronizing the databases by “master slave replication”.

If a database fails then the subsequent request will be forwarded to the other database in network and since they were synchronised their content remains same.

Even if database fails while inserting/updating/deleting an entry, it won’t be able to send any reply to terminal server. In that case terminal server will time out and will try to make changes into database via another server.Here also no data is lost and is transparent to client.

**Chapter 8**

**Testing and Results of Fault Tolerant Approach**

So far we have described about the model and its implementation. Here we deal with the fidelity of the system.

So the testing of system is as follows.

As the prior most step we execute all the programs on the machines.First run the main server code followed by terminal servers and client.Client is all set to send request and servers are ready to carry their functions.

The only point of failure that needs testing in our approach is when a terminal server goes down.

As mentioned from previous chapter the point of failure at point 3 is tested .

We have continuously sent the request from the client unaware of the availability of terminal server.

In day to day work, failure occurs due to power cut , network disconnection or overload on server.

So we preferred network disconnection as the source for creating a mishap/fault on terminal server.

When the server is brought down (disconnection network access), within a span of seconds all the incoming requests to it are redirected towards another terminal server by the main server and the result is displayed back on client terminal.Client gets the desired result .

Like wise we bring down other terminal servers (either the one computing or the one available with an idle status) and test the failure.It is noted that client requests are attended by other available servers.

Now when the terminal server is back or revives into the system it is treated afresh as available with idle status and hence can be assigned to cater client requests.

**Chapter 9**

**SCOPE AND FUTURE IMPROVEMENTS**

No system can be made cent percent fault tolerant, hence there is also some scope of improvement.

In our case main server which basically acts as a load balancer is main bottleneck.

So the first improvement can be on this. In case of failure of main server, since we are using IP address for connecting clients to main server, whole model goes down and it becomes the single point of failure. To deal with this we can have a name implementation ie in place of using IP to connect we can use a name for the connection and a DNS server can keep the mapping of name to IP address. In case of failure any of the terminal server can be chosen to act as a main server via some leader election process and then the new main server just need to provide its IP to DNS server for mapping with corresponding name.Now one of the terminal server will act as a main server and cater to the incoming requests.

Since every new request from a client can go to any available server, we can try to make all requests from a client to be served by a particular terminal server until it fails. This will be more appropriate as now each terminal server needs to maintain records of clients connected to it and clients transferred to it in case of failure of other terminal server.

We can reduce the number of context switch between terminal servers and databases for entering a record with the help of local file or data structure at terminal server.All the changes are made to this file first and then later after some given interval all the changes are made to the database.

**Chapter 10**

**REFERENCES:**

[1] Falko Timme, How To Set Up Database Replication in MySQL (01/14/2006),

[web page] http://www.howtoforge.com/mysql\_database\_replication.

[2] Brian Selic, Staff, IBM, Software Group ,Fault tolerance techniques for distributed systems

(27 Jul 2004), [web page] http://www.ibm.com/developerworks/rational/library/114.html

[3] A. Avizienis, "Architecture of fault-tolerant computing systems," in Dig. 1975 Int. Symp. Fault-Tolerant Computing, Paris, France, June 1975.