**Development of an efficient multiple client-server model for a medical application using load balancing technique**

Ambika Pagavathiraj

Devi Kandasamy

Preethi Yellappa

Table of Contents

1. Table of Contents---------------------------------------------------------------------------------------------2

2. Introduction---------------------------------------------------------------------------------------------------4

Objective

What is the problem?

Why this is a project related the this class

Why other approach is no good

Why you think your approach is better

Statement of the problem

Area or Scope of investigation

3. Theoretical bases and literature review----------------------------------------------------------------5

Definition of the problem

Theoretical background of the problem

Related research to solve the problem

Advantage/Disadvantage of those research

Your solution to solve this problem

Where your solution different from others

Why your solution is better

4. Hypothesis---------------------------------------------------------------------------------------------------7

5. Methodology-------------------------------------------------------------------------------------------------7

How to generate/collect input data

How to solve the problem

Algorithm design

Language used

Tools used

How to generate output

How to test against hypothesis

How to proof correctness

6. Implementation-----------------------------------------------------------------------------------------8

Design Diagram

Flowchart

7. Data analysis and discussion -------------------------------------------------------------------------11

Output generation and analysis

Compare output against hypothesis

8. Conclusions and recommendations------------------------------------------------------------------13

Summary and conclusions

Recommendations for future studies

9. Bibliography------------------------------------------------------------------------------------------------14

10. Appendices------------------------------------------------------------------------------------------------15

Program flowchart

Program source code with documentation

2. Introduction

The client-server model is an approach to computer network programming. Clients and servers exchange messages in a request response pattern. To communicate the client and server have specific languages and rules and they are defined as communication protocols and all these protocols operate in the application layer.

Client-server has several design alternatives. In this project, an concurrent server is used. A Concurrent server handles many clients at a time by spawning itself which has been proven efficient compared to other types of servers. Also techniques of load balancing is applied at the server end.

Load balancing is a computer networking concept to distribute workload across multiple computers to achieve optimal resource utilization, maximize throughput, minimize response time and avoid overload.

**Objective**

The objective of our project is to efficiently implement multiple clients and two server and load balancing concept for a class of Health Information System.

In general, the objective of the project is to create a multiple server and client communication with an effective load balancer. The communication allows multiple clients (nearly 1000) to connect to two servers. Our goal is to come up with a solution to improve on the basic client-server technology.

**What is the problem?**

There are several problems relating to the client server model:

1. Increasing users increases the scalability problems (e.g. Delivering server data and information to large number of clients)
2. Unavailability of information or data when a server crashes.
3. Eavesdropping and tampering of data during communication.

**Why this is a project related to this class?**

Our Computer networks class basically deals with the study of layers in OSI model and our project deals with the application layer and the transport layer. Also client-server model is an approach to computer network programming.

**Why other approach is no good?**

1. Iterative servers: Handles only one client at a time. It is not very efficient with healthcare applications.
2. Client- Tiered Server cluster: each server might be accessed by others. It can slow down the performance of the server.
3. In connectionless communication there is no dedicated connection between the host and client computer. So there is no reliability.

**Why you think your approach is better?**

1. Concurrent server: Handles many clients at a time. It can process many client requests at a time without having to make the client wait.
2. Load balancing technique: it improves the overall performance of the application by decreasing the burden on servers.
3. Connection-oriented communication involves a dedicated connection between host and the client computer, which means if there is an error in the receiver side the information can be resent.

**Statement of the problem**

The main issue with client server model is handling of multiple clients and servers.

3. Theoretical bases and literature review

**Definition of the problem**

Client requests may not be processed when:

1. The number of users goes up, an obvious decrease in performance of server can be noticed.
2. Also if there is only one server handling multiple clients, and if the server crashes

**Theoretical background of the problem**

Increasing users increases the scalability problems (e.g. Delivering server data and information to large number of clients. Unavailability of information or data when a server crashes.

**Related research to solve the problem**

The papers that were referred to address these problems are:

1. IPv4 and IPv6 Client-Server Designs: The Sockets Performance.
2. The Analysis of the Basic Model of the Multilayer Client-Server Queuing Network.
3. Extension of client server applications to the internet.
4. Integrating Security Considerations in Client Server Architectures of Health Information Systems Development

**Advantage/Disadvantage of those research**

The Advantages of these are:

- They provide details of the concurrent server approach.

- Use of two concurrent servers instead of client tiered server cluster.

- Security requirements.

The Disadvantages are:

- They do not tell us how to effectively handle the client incase both the servers crash.

- Ideas are limited.

**Your solution to solve this problem**

Our approach is a step by step method to solve the problem.

Step 1) One server, multi clients and a database.

Step 2) One server with one databases, a load balancer and multiple clients.

Step 3) Two servers with one database, one load balancer and multiple clients.

**Where your solution is different from others**

There have been applications of multiple servers using peer to peer technology. But our solution revolves around load balancing technique.

**Why your solution is better**

Our approach can handle multiple client requests.

Load balancing technique improves the overall performance of the application by decreasing the burden on servers. Connection-oriented communication involves a dedicated connection between host and the client computer, which means if there is an error in the receiver side the information can be resent.

4. Hypothesis

Using our approach, client server communication goes on even if one of the servers crash.

5. Methodology

**How to generate/collect input data -**

The servers wait for the connection and the load balancer verifies the status of the servers. The client gives input in form of the symptoms to know the risk of diseases.

**How to solve the problem -**

This section presents a summary of one of the methods for efficient client-server communication.

Step 1) One server, multi clients and a database.

Step 2) Two servers with two databases, a load balancer and multiple clients.

Step 3) Two servers with one database, a load balancer and multiple clients.

**Algorithm Design**

1. **LoadBalancer :**

Main thread spawns “server intro thread” and “status checker thread”

* **Server Intro thread:** Listens for new server connections.
* **Status Checker**: Sleeps and looks up the server registered with it on the server connection table.

1. **When Server connects:**

* Main thread sends its ip, listen port no and status port no to Load Balancer’s Server Intro thread.
* Main thread spawns to the status thread and the status thread listens for the ‘Load Balancer’s status checker thread to connect to server status thread.
* Waits for connection from LoadBalancer.
* Server Intro thread on Load Balancer receives ip, server listen port, server status port information and adds it to server connection table

1. **When Client connects:**

* Client connects to load balancers main thread listen port. Main thread Load Balancer uses scheduling algorithm such as round robin to pick the next server socket from server connection table.
* Main thread on load balancer spawns connection thread
* Connection thread on load balancer connects to chosen server listen port
* Next main thread on server spawns the connection thread after accept. [connection complete]
* Main thread on LB & sever continue to listen for new connections.

1. **Server failure scenario:**

* Server goes down
* Status checker thread on LB fails to connect to the server
* It gets removed from the connection table entry

**Language used :**

Java

**Tools used**

Eclipse, MySQL

**How to generate output-**

Start the server, client and load balancer processes and the output generated is Patients Information.

**How to test against hypothesis -**

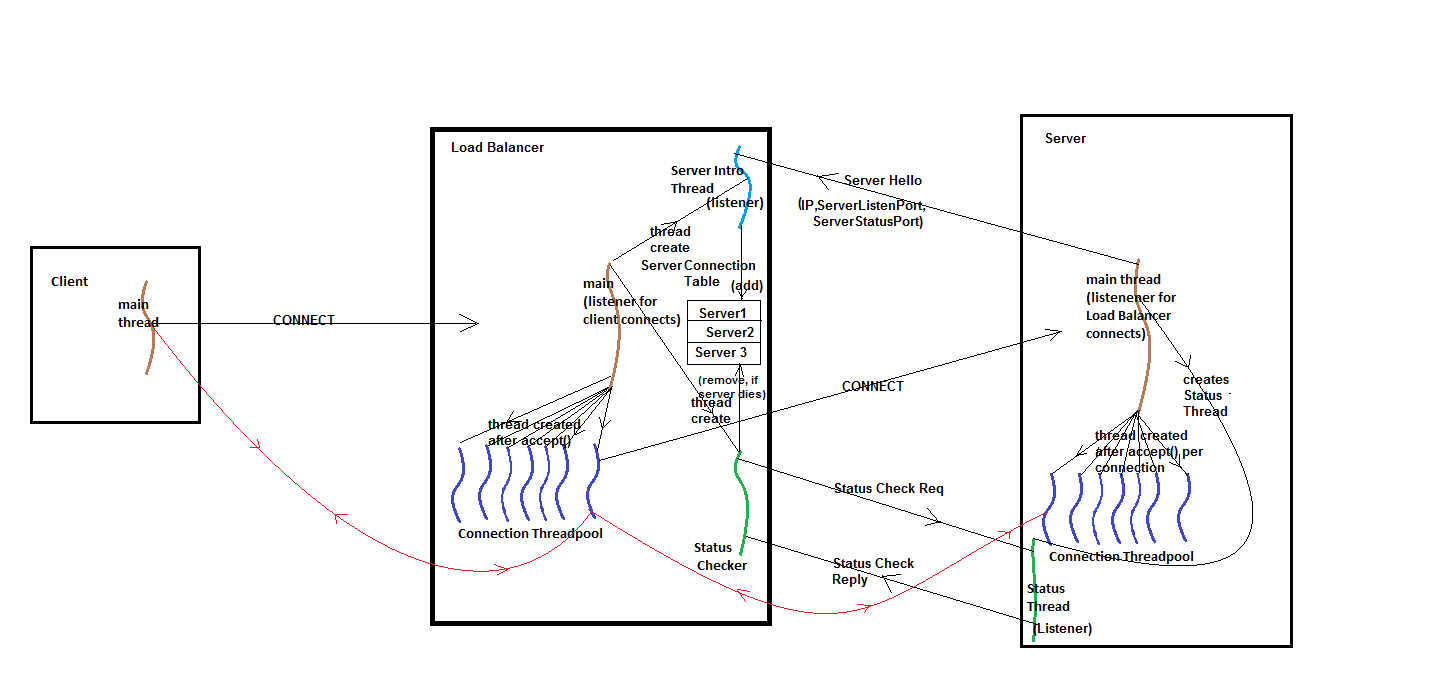
In case one of the servers goes down, the load balancer must be able to detect this and reroute the client requests to the available server. Also when the both the servers are down, the load balancer must be able to send an error message to the clients.

**How to proof correctness**

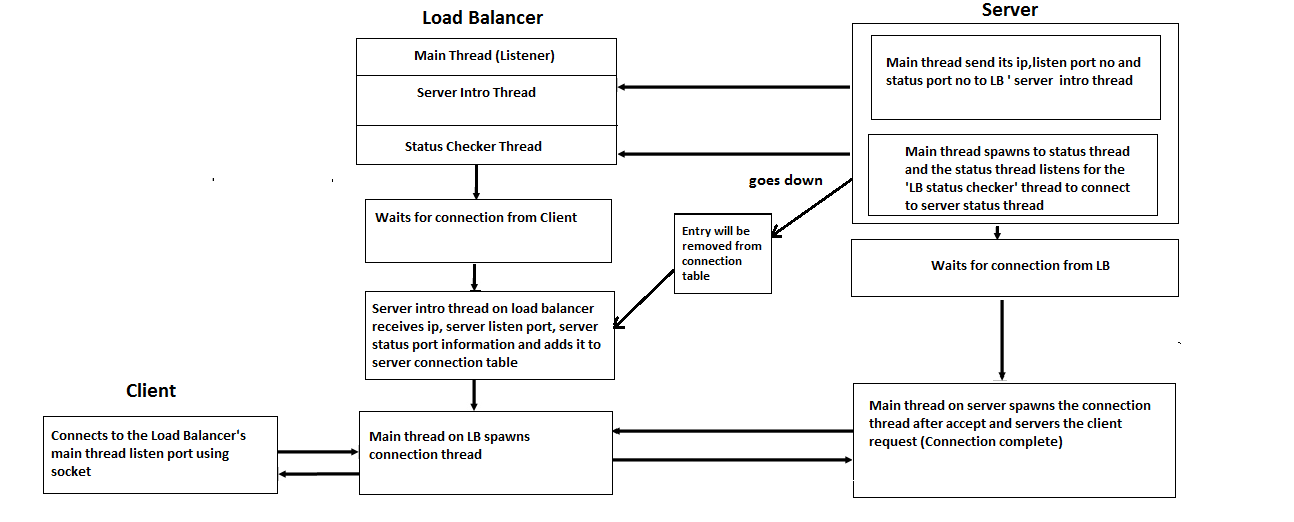
Multiple client requests are handled by server.

6. Implementation

**Design Diagram:**

****

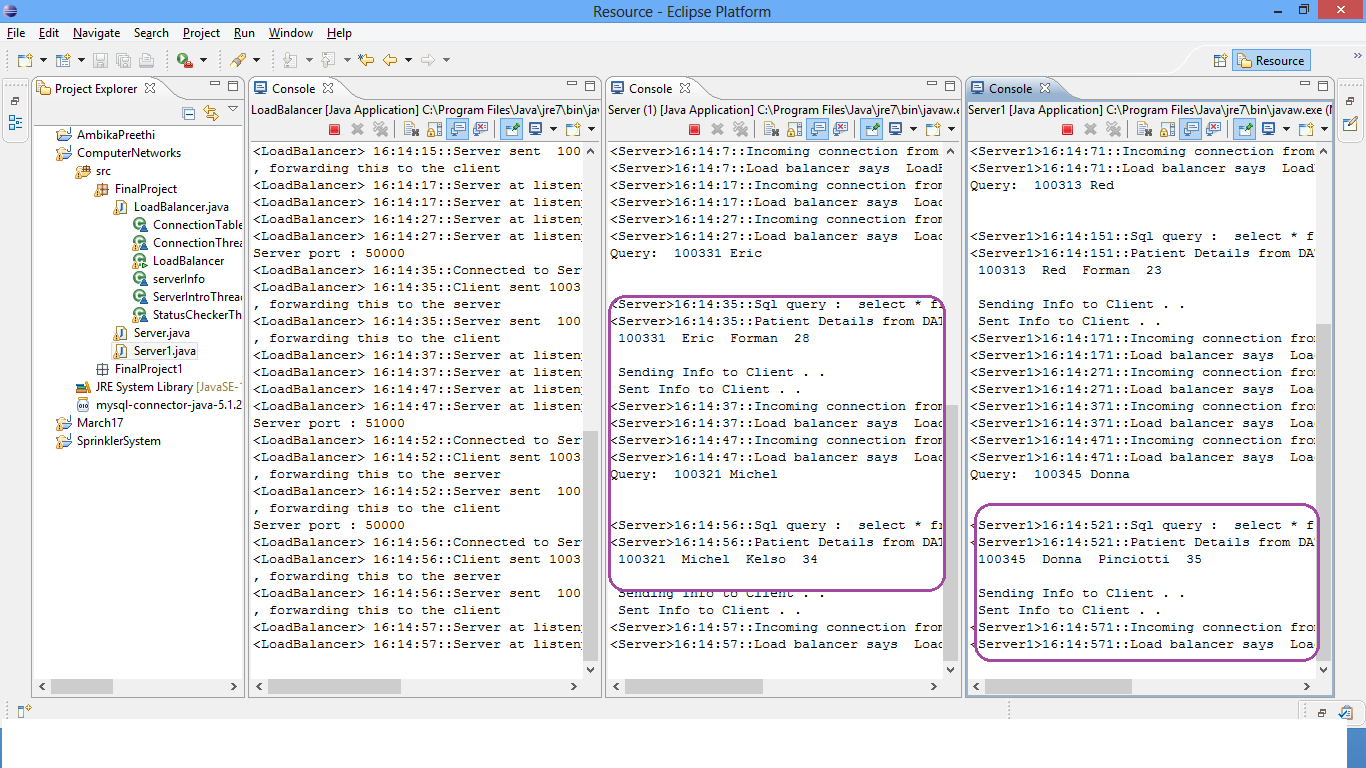
**FlowChart:**

****

7. Data analysis and Discussion

**Output Generation and analysis:**

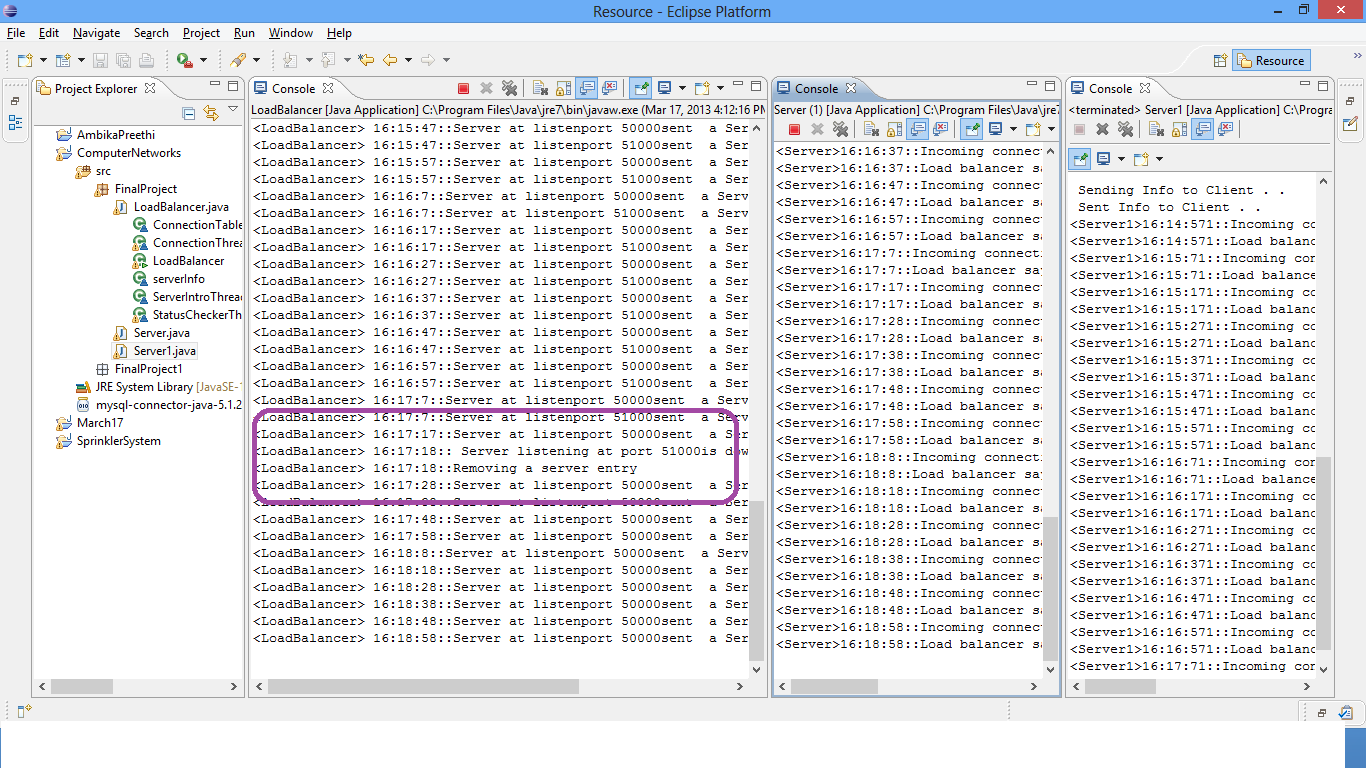
**Server 1 and server 2 handling the client request**

****

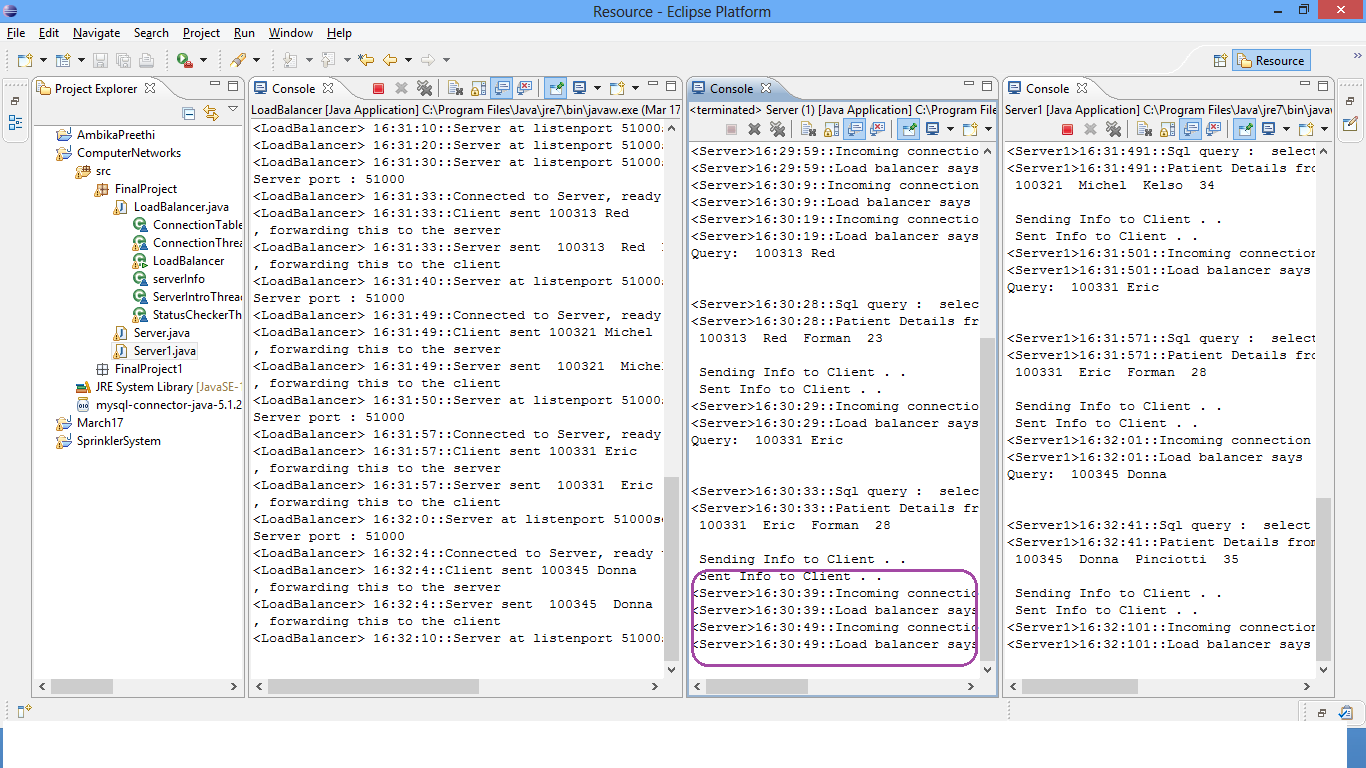
**Abnormal case explanation:**

**Output after Server1 goes down**

**Entry from connection table is removed**

****

**Since Server 1 is down, Server 2 is handling the Client request**

****

8. Conclusions and Recommendations:

**Summary and conclusions:**

Client server model is an approach to network programming. This project was implemented in a step by step approach to come up with an efficient model which is centered around the concepts of the load balancer and is also scalable.

**Recommendations for future studies:**

Backup load balancer:

In scenarios when the load balancer fails, the backup load balancer takes charge.

**Backup database:**

Since all the information is stored in the database, a backup database is necessary in case the first one fails.

9. Bibliography

[1] IPv4 and IPv6 Client-Server Designs: The Sockets Performance (2012)

Teddy Mantoro, Media A. Ayu, Amir Borovac and Aqqiela Z. Z. Zay Department of Computer Science, KICT International Islamic University Malaysia, Kuala Lumpur, Malaysia.

[2] Integrating Security Considerations in Client Server Architectures of Health Information Systems Development

Chih-Cheng Lien Chang-Wu Liu YI-An Chen

Department of Computer Science and Information Management

Soochow University, Taipei, Taiwan

[3] http://en.wikipedia.org/wiki/Load\_balancing\_(computing)

[4] The Application Reseach of Mixed Program Structure based on Client-Server, Browser-Server and Web Service

Jian-fei Tu, Rui-feng Guo

[5] Extension of Client-Server Applications to the internet

Duo Li, Yoshizumi Serizawa, Senior Member, IEEE, Mai Kiuchi

10. Appendices:

**Code:**

****

****

****

**Program Flow chart:**

