BHARATIYA VIDYA BHAVAN'S

SARDAR PATEL INSTITUTE OF TECHNOLOGY

(Autonomous Institute Affiliated to University of Mumbai)

MUNSHI NAGAR, ANDHERI (WEST), MUMBAI – 400 058

Department of Computer Engineering 2022-2023

Distributed Banking System

Class: TE COMPS

Name of Members:

Aakanksha Dhawale(2020300008)

Sakshi Gaikwad (2020300011)

Chirag Girdhar (2020300013)

Shubham Golwal (2020300015)

Abstract

An online banking system is a replication of a banking system which is held over the internet. Just like a bank it allows users to transfer money to other accounts and check balances. It is developed with the objective of making the system reliable, easier and fast. The work presented here develops an online distributed banking system and proposes an implementation, keeping in mind the scalability and integrity of the system.

Introduction

An online banking system is a mechanism that handles transactions over the internet and reduces the paperwork involved. With increase in technological development, online banking systems help reduce time in transactions.

Since an online banking system has to allow for remote clients present all over the world to perform transactions, it raises the need for a distributed system. Such a system must allow for low response time, high availability and be able to support a large number of users at the same time. It is a very real time system, where time precision to the second matters. Thus a robust distributed architecture is the requirement.

In this project, we have created a distributed banking system that satisfies the requirements stated above by utilizing various algorithms to ensure server load balancing, data consistency, mutual exclusion and clock synchronization.

Objectives

- To develop a client-server based Online Banking Distributed System with multipleservers and clients.
- To establish a communication channel between client and remote server. ● To implement Clock Synchronization of servers and clients to ensure all transactions are recorded with the synchronized timestamp.
- To prevent race conditions and maintain process synchronization among servers usingthe Mutual Exclusion algorithm.
- To maintain consistency of data with respect to transactions of clients.
- To prevent any server from being overwhelmed by Load Balancing.

Architecture

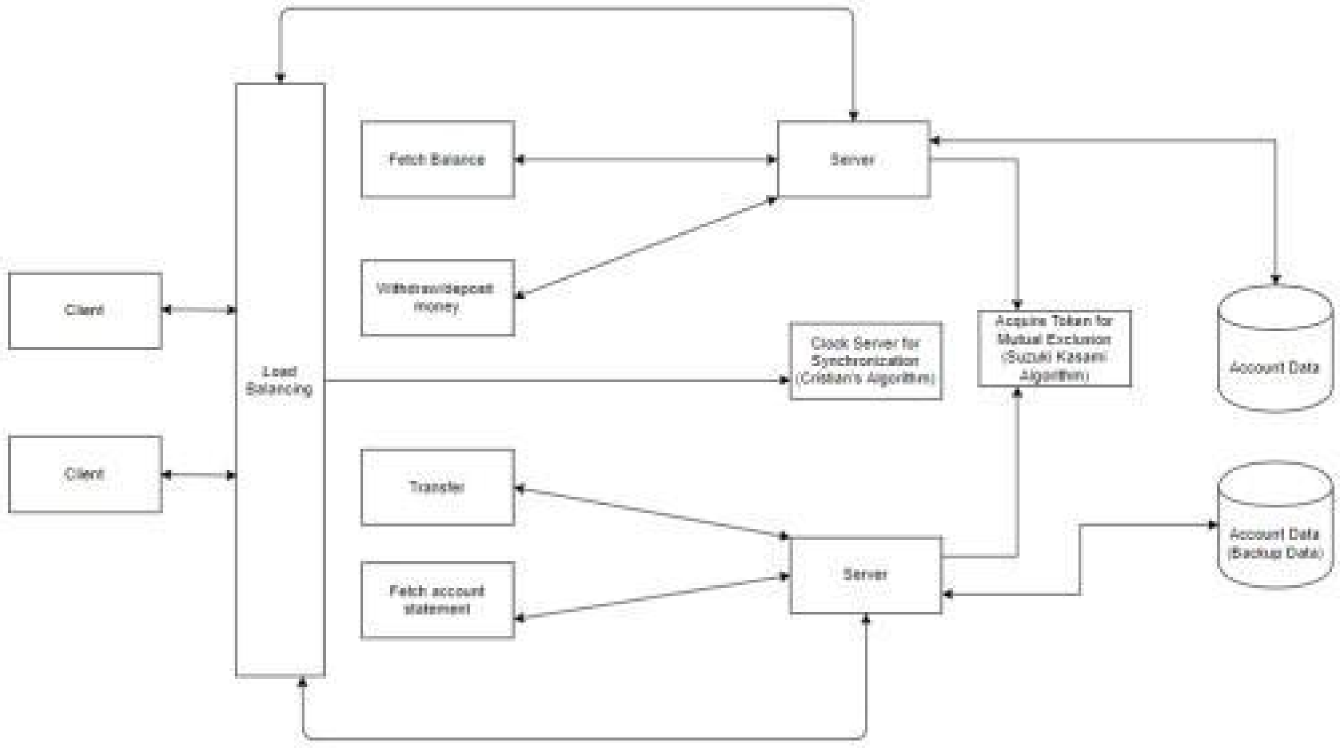

Client
Client
Load Balancing
Fetch Balance
Withdraw/deposit money
Transfer
Fetch account statement
Server
Server
Clock Server for Synchronization (Cristian's Algorithm)
Acquire Token for Mutual Exclusion (Suzuki Kasami Algorithm)
Account Data
Account Data (Backup Data)

Fig. 1: System Architecture Diagram

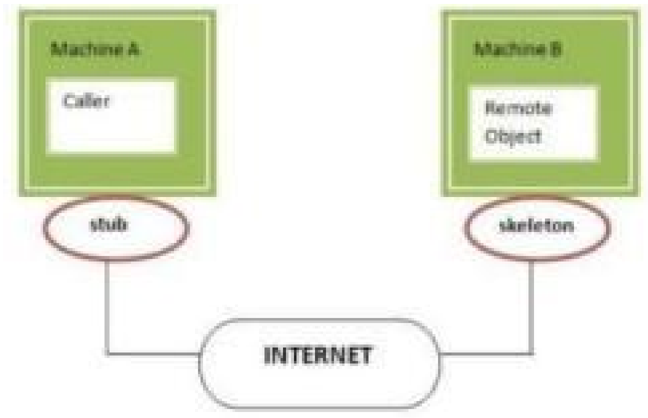
Fig. 1 shows the system architecture of the Distributed BankingSystem. The system allows support for multiple clients and consists of 3 servers.

For each client that wishes to use the system, their request is passed through a Load Balancer first. This load balancer handles requests using the Round Robin algorithm. Each client wishing to make a request first synchronizes it's clock by using the clock server applying Cristian's algorithm. Each server that receives a money transfer request, waits to obtain the token based lock on the shared resource - account information datastore. The token lock is acquired and released via the Suzuki Kasami Algorithm. A copy of the primary datastore (backup) is created with strict consistency to allow for data consistency and recovery.

Methodology

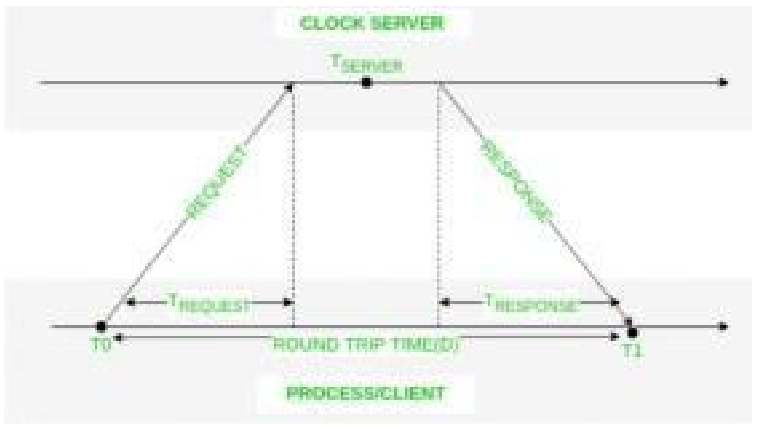
1) Client-Server Communications

The first step was to create a client server based program using RMI. Remote Method Invocation (RMI) is an API which allows an object to invoke a method on an object that exists in another address space, which could be on the same machine or on a remote machine.



2) Clock Synchronization

Next step, we implement a clock synchronization algorithm for a banking system. We used Cristian's algorithm for clock synchronization. Cristian's Algorithm is a clock synchronization algorithm that is used to synchronize time with a time server by client processes. This algorithm works well with low-latency networks where Round Trip Time is short as compared to accuracy while redundancy prone distributed systems/applications do not go hand in hand with this algorithm. Here Round Trip Time refers to the time duration between start of a Request and end of corresponding Response.



3) Mutual Exclusion

In order to prevent race conditions, we implemented a mutual exclusion algorithm for a banking system. Mutual exclusion is a concurrency control property that is introduced to prevent race conditions. It is the requirement that a process can not enter its critical section while another concurrent process is currently present or executing in its critical section i.e only one process is allowed to execute the critical section at any given instance of time.

We implemented Suzuki Kasami algorithm which is a token based algorithm for achieving mutual exclusion in distributed systems. This is modification of Ricart–Agrawal algorithm, a permission based (Non-token based) algorithm which uses REQUEST and REPLY messages to ensure mutual exclusion.

4) Load Balancing

Load balancing refers to efficiently distributing incoming network traffic across a group of backend servers, also known as a server farm or server pool. Modern high‑traffic websites must serve hundreds of thousands, if not millions, of concurrent requests from users or clients and return the correct text, images, video, or application data, all in a fast and reliable manner. To cost‑effectively scale to meet these high volumes, modern computing best practice generally requires adding more servers. A load balancer acts as the “traffic cop” sitting in front of your servers and routing client requests across all servers capable of fulfilling those requests in a manner that maximizes speed and capacity utilization and ensures that no one server is overworked, which could degrade performance. If a single server goes down, the load balancer redirects traffic to the remaining online servers. When a new server is added to the server group, the load balancer automatically starts to send requests to it.

We implemented the Round Robin method for load balancing since in this the requests are distributed across the group of servers sequentially. For example, the first request gets the IP address of server 1, the second request gets the IP address of server 2, and so forth, with requests starting again at server 1 when all servers have been assigned an access request during a cycle.

5) Data Consistency

Final step, we implemented Data Consistency using Java. Consistency in database systems refers to the requirement that any given database transactions must change affected data only in allowed ways. Any data written to the database must be valid according to all defined rules, including constraints, cascades, triggers, and any combination thereof. This does not guarantee correctness of the transaction in all ways the application programmer might have wanted (that is the responsibility of application-level code) but merely that any programming errors cannot result in the violation of any defined database constraints. Data Consistency refers to the usability of data. Data Consistency problems may arise even in a single-sit environment during recovery situations when backup copies of the production data are used in place of the original data. In order to ensure that your backup data is usable, it is necessary to understand the backup methodologies that are in place as well as how the primary data is created and accessed. Another very important consideration is the consistency of the data once the recovery has been completed and the application is ready to begin processing

Outcomes

- Clients are successfully able to connect to the server using Remote Method Invocation (RMI).
- The clocks of multiple servers and clients are synchronized using Cristian's Algorithm before making requests to the server.
- The process synchronization is implemented among servers using the Suzuki MassMutual Exclusion Algorithm.
- The load on multiple servers is managed by using Round Robin Load Balancing Algorithm.
- The strict consistency database is maintained using data replication.

Conclusion

- Through this project, we were able to successfully implement a distributed system for an Online Banking System.
- The Online Banking System provides functions like view balance and crediting and debiting money in different accounts.
- We have successfully implemented various distributed features like RMI, Clock Synchronization, Load Balancing, Mutual Exclusion and Data Consistency and replication.
- Using this, we were able to create a scalable server that could handle multiple transactions concurrently.