FAULT TOLERANT CHAT SERVER SYSTEM DESIGN DOCUMENT

SYSTEM DESIGN DOCUMENT

Overview
To make a chat room using gRPC and Protocol buffers wherein users can follow other user and get messages (status updates) from other users that they follow. Users may also send their own updates to all the others following them. The system should be fault-tolerant, highly available, and scalable.

1 INTRODUCTION

1.1 Purpose and Scope

To be able to design and develop highly scalable and fault-tolerant distributed system design by applying concepts like mutual exclusion, leader election, consensus and data replication among different server nodes.

1.1.1 System Overview

This system is similar to status update on FB or Twitter. Each user has his/her own chat-room and they may post their updates in their chat-room. If one user follows any other user, he/she will get updates when other users post in their chatroom. User may join or leave the chat-room or can also post an update.

1.1.2 Design Constraints

To be able to receive updates of other users, a user must be following other user's chat-room. When a user wants to post any update he/she must be in chat mode. When in chat mode the user cannot do any alteration to joining or leaving other users' chat-rooms. User must exit from chat mode to be able to join or leave other chat-rooms.

Things to be taken care to make system fault-tolerant
- Server process can be killed randomly. Server should be able to come up within a specified time limit (30 seconds) to serve the future client requests.
- Server machines can be shut down (means all processes will be killed from that particular machine) except the master machine, though processes in aster machine can be killed.
- Network Interfaces can be shut down
- Systems can be in different date time settings. The order of the chats should be consistent among all the nodes

1.2 Project References

GRPC and Google Protocol buffers.

2 SYSTEM ARCHITECTURE

2.1 System Software Architecture

Server
1) Server will always be up listening to clients for requests of creation, joining, leaving or chatting.

2) It will also store information of all chatrooms in local file system.
3) It will store the last 20 messages that have been received and will send those to every client entering chat mode.

Client
1) Client when connects to server, will by default create it's own chatroom. Then others can join the new client's chatroom.
2) Client when joins other client's chat-room, it will keep polling server to check if any new updates have been received on the chat-rooms it is following.
3) Client can send updates from his chat-room to all his followers.
4) Client can anytime leave the chat mode by using exit command.
5) Client can disconnect anytime using Leave command.

3 HUMAN-MACHINE INTERFACE

3.1 Inputs

3 startup scripts needs to be run in 3 different machines to make all the server processes up. One for the master machine and rest 2 scripts for the replica machines.

Once the scripts are run successfully, clients can be started using the commands
./fbc -u <username>

3.2 Outputs

Updates on activity going in all chat-rooms in form of messages and join/leave notification.

4 DETAILED DESIGN

4.1 Software Detailed Design

The major component of the systems are

- **Client Program:** This program connects to the server for the chat operation

- **Master Server process** (Runs on the master machine): This is the primary master process which keeps the track of all worker processes and the master replica processes with regular heartbeats

- **Master Replica Process** (Runs on the master machine): These processes are there to serve when the Master process will be killed. One of these processes will become the leader and wake up the killed process.

- **Worker Process** (Runs on all the machines): These are the actual processes which serves the requests of the clients. If one worker process is killed, master process will get to know through the regular heartbeat update and wake up the killed process.

FAULT TOLERANT CHAT SERVER SYSTEM DESIGN DOCUMENT

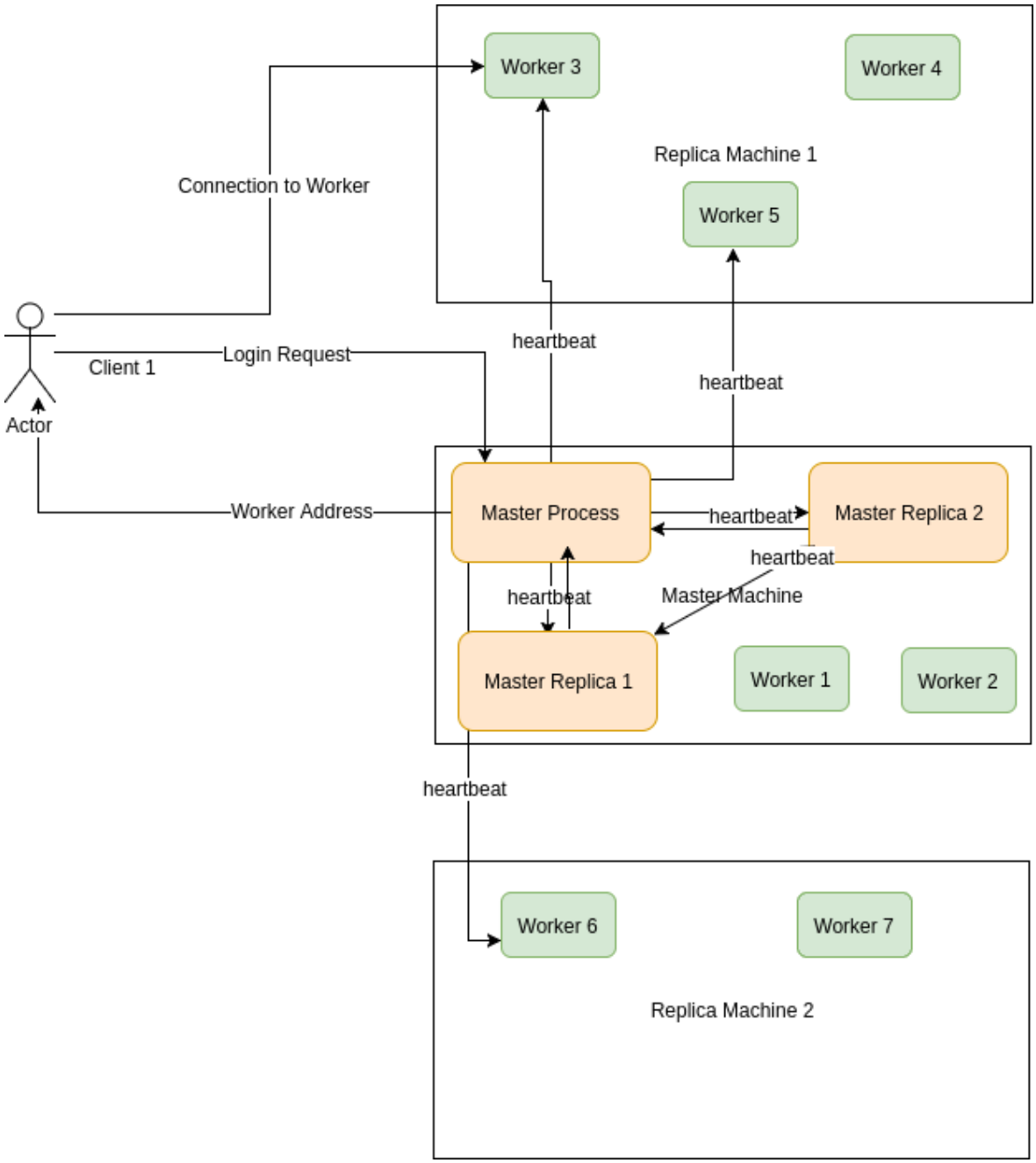

Fig 1: High Level Design System Diagram