Fault tolerant key-value server/client using MPI

Distributed System

Group members

Nguyen Xuan Tung	BI10-188
Nguyen Quang Anh	BI10-012
Lu Khanh Huyen	BI10-083
Vu Duc Chinh	BI10-024
Tran Hong Quan	BI10-149

Table of contents

01
Introduction

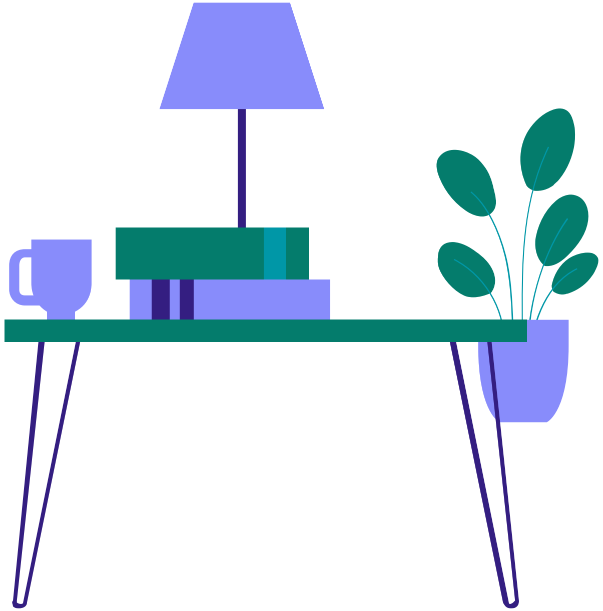
Why do we need this

- High performance distributed system = large number of nodes ⇒ increase possibility of failures.
- Fault tolerance(FT) system is important.

Fault-tolerance

Fault-tolerance is the ability of the system to "survive" and continue operating in the face of failures.

The failure varies from network failure, hardware infrastructure failure, and software failures.



Key-value database

- A key-value(KV) database is a type of database that stores data in key-value format.
- The key is unique and is mapped to a specific value.
- Operations in KV database: add, update, delete, query.

Redis

- Redis stands for Remote Dictionary Server
- Redis supports different types of values such as list, set, hash, sorted set.
- Redis can be used on top of Node-JS to speed up the query process in web applications.

Message-passing Interface (MPI)

- MPI is a library designed to create programs that run efficiently on parallel architectures.

02

Objective

Goal

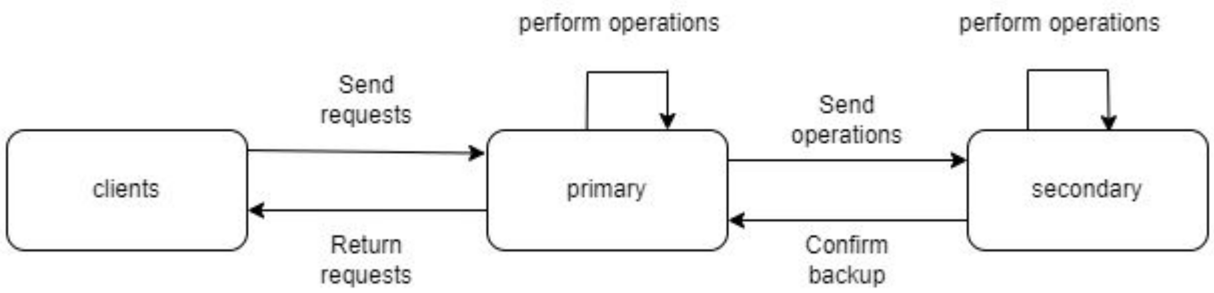
- To simulate a primary/secondary server that can support fault-tolerance.

03

Methodology

Primary/Secondary Replication

- Primary/Secondary Replication is a mechanism where the primary replica forwards the operations sent from the client to the secondary replica to achieve the same state.



Normal Scenarios

- The client sends a request to the primary server(e.g., GET/PUT/DELETE).
- If the client sends a GET request, the primary server will return its corresponding value to the client.
- If the client sends a PUT/DELETE request, the primary server will add/delete the key-value data to its database and forward the data and the operation to the secondary server.
- The secondary server performs the same operation as the primary server and sends a acknowledgement message to the primary server.
- The primary server will send a finished PUT/DELETE message to the client.

Failure Scenario

- Implement a random function to simulate the case of failure.
- Probability of failure set to 20%.
- Primary server becomes back-up server and vice versa. Client will now connect to the new primary server.

04

Demo

05

Conclusion and Future work

Conclusion

- In this project, we have applied MPI to create one client and two servers. We also successfully implemented primary/secondary replication, the simplest way of replication.

Future work

- Implement several backup servers instead of one backup server.
- Implement a metadata server that can manage multiple clients/servers interactions.
- We also do not implement failure scenarios such as network failure or machine crash.