Fault-Tolerant Key-Value Server/Client Using MPI (Redis Clone)

Group 2

December 23, 2024

Contents

1 Introduction

Distributed systems require robust fault-tolerance mechanisms to ensure high availability and reliability. This project implements a simplified fault-tolerant key-value store similar to Redis using the Message Passing Interface (MPI). The system features client-server communication, data replication, and checkpointing to ensure fault tolerance.

The primary goals of this project are:

- Implement a key-value store with basic operations: `SET`, `GET`, `DELETE`.
- Achieve fault tolerance through data replication and checkpointing.
- Demonstrate failure recovery with backup servers.

2 System Design

2.1 Key Components

The project consists of the following components:

- **Primary Server**: Handles client requests and maintains the main key-value store.
- **Backup Servers**: Replicate data from the primary server and act as a fallback in case of failure.
- **Clients**: Send commands to the server and retrieve responses.

2.2 Fault-Tolerance Mechanisms

1. **Replication**: The primary server replicates its state to backup servers using MPI.
2. **Checkpointing**: The key-value store is periodically saved to disk to recover from unexpected failures.

2.3 Communication Model

The system uses MPI functions for inter-process communication:

- `MPI_Send`: To send messages (commands or data).
- `MPI_Recv`: To receive messages.

3 Implementation

3.1 Server Code

The server handles requests from clients and manages replication and checkpointing. Below is the implementation:

```python
from mpi4py import MPI
import json

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

PRIMARY_SERVER = 0
BACKUP_SERVERS = list(range(1, size))
kv_store = {}

def save_checkpoint():
    with open(f"checkpoint_{rank}.json", "w") as f:
        json.dump(kv_store, f)

def load_checkpoint():
    try:
        with open(f"checkpoint_{rank}.json", "r") as f:
            return json.load(f)
    except FileNotFoundError:
        return {}

def replicate_to_backups():
    for backup in BACKUP_SERVERS:
        comm.send(kv_store, dest=backup, tag=1)

if rank == PRIMARY_SERVER:
    kv_store = load_checkpoint()
    while True:
        command = comm.recv(source=MPI.ANY_SOURCE, tag=0)
        action = command["action"]
        key = command.get("key")
        value = command.get("value")

        if action == "SET":
            kv_store[key] = value
            replicate_to_backups()
            save_checkpoint()
            comm.send({"status": "OK"}, dest=command["client_rank"])
        elif action == "GET":
            result = kv_store.get(key, "Key not found")
            comm.send({"status": "OK", "value": result}, dest=command["
                client_rank"])
        elif action == "DELETE":
            if key in kv_store:
                del kv_store[key]
                replicate_to_backups()
                save_checkpoint()
                comm.send({"status": "OK"}, dest=command["client_rank"
                    ])
            else:
                comm.send({"status": "Key not found"}, dest=command["
                    client_rank"])
else:
    while True:
        kv_store = comm.recv(source=PRIMARY_SERVER, tag=1)
```

Listing 1: Server Code

3.2 Client Code

Clients send commands to the server and display responses. Below is the implementation:

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

def send_command(action, key=None, value=None):
    command = {
        "action": action,
        "key": key,
        "value": value,
        "client_rank": rank,
    }
    comm.send(command, dest=0, tag=0)
    response = comm.recv(source=0)
    return response

if __name__ == "__main__":
    while True:
        user_input = input("Enter command (e.g., SET key value): ").
            strip().split()
        if not user_input:
            continue
        action = user_input[0].upper()
        if action == "SET" and len(user_input) == 3:
            key, value = user_input[1], user_input[2]
            print(send_command(action="SET", key=key, value=value))
        elif action == "GET" and len(user_input) == 2:
            key = user_input[1]
            print(send_command(action="GET", key=key))
        elif action == "DELETE" and len(user_input) == 2:
            key = user_input[1]
            print(send_command(action="DELETE", key=key))
        elif action == "EXIT":
            break
        else:
            print("Invalid command.")
```

Listing 2: Client Code

4 Testing

4.1 Basic Functionality

We tested the basic operations of the system:
`SET`, `GET`, `DELETE`. Commands work as expected, and responses are received correctly.

4.2 Fault Tolerance

- Simulated failure of the primary server by killing its process.
- Verified that backup servers maintained the latest state through replication.

5 Conclusion

This project demonstrated a fault-tolerant key-value server/client system using MPI. Key features such as replication, checkpointing, and failure recovery were successfully implemented. Future improvements could include strong consistency mechanisms and scalable partitioning.

6 Roles

Contributed to this project:

- Le Nhat Anh (22BI13018): Leader
- Le Tuan Anh (BA11-005): System design
- Vu Ngoc Minh (BA12-128): Implementation
- Nguyen Ngoc Anh Duc (22BI13093): Testing
- Ha Tan Minh (BA12-126): SETUP script
- Le Phuc Nguyen (22BI13338): Write report in LaTeX.