RAFT consensus algorithm over a distributed Key-Value store

Kalopisis Ioannis 7115112200011
Notaris Ioannis 7115112200038

June 29, 2023

Contents

1 Introduction

Distributed systems have gained immense popularity in recent years due to their ability to handle large-scale data storage and processing. Key-value stores, a fundamental building block of many distributed systems, provide a simple and efficient way to store and retrieve data based on unique keys. In this report, we present the design and implementation of a key-value store that utilizes the Raft consensus algorithm for fault-tolerant replication and consistency across a cluster of nodes.

The key-value store we have developed serves as a reliable and scalable solution for storing and retrieving key-value pairs in a distributed environment. The system is designed to handle high volumes of read and write requests while ensuring strong consistency guarantees. By leveraging the Raft consensus algorithm, our system achieves fault tolerance and replication, enabling it to operate reliably even in the face of node failures or network partitions.

The core functionality of our system revolves around the interaction between a client program, the key-value nodes, and the Raft algorithm. Clients send messages to the key-value store, which then forwards them to the Raft consensus algorithm. Raft ensures that the messages are replicated across a specified number of nodes in the cluster, maintaining consistency and fault tolerance. Once a response is obtained from the key-value nodes, it is sent back to the client program. This process enables clients to interact seamlessly with the key-value store, enjoying the benefits of fault tolerance, scalability, and consistency provided by the underlying Raft algorithm.

In the following sections of this report, we will delve deeper into the design choices and implementation details of our key-value store and the integration of the Raft consensus algorithm. We will explore the architecture, communication protocols, and fault-tolerance mechanisms employed in the system. Ultimately, our goal is to present a comprehensive overview of our key-value store and Raft implementation, showcasing the robustness and efficiency of our distributed system.

2 External Client

3 Key-Value store

The Key-Value Store is a distributed server-based application that facilitates the storage and retrieval of data using unique key-value pairs. It provides a simple yet powerful abstraction for organizing and accessing data efficiently. This section provides a comprehensive overview of the Key-Value Store application, including its purpose, architecture, functionality, advantages, and potential use cases.

The purpose of the Key-Value Store application is to offer a scalable and efficient solution for data storage and retrieval. By using a key-value pair approach, it enables clients to easily store and retrieve data using unique keys.

3.1 Architecture

The Key-Value Store application follows a distributed architecture that ensures high availability, fault tolerance, and scalability. Data is distributed and replicated across multiple servers, with each server responsible for a specific portion of the key space. Clients interact with the servers to perform operations like storing, retrieving, updating, and deleting key-value pairs. The servers communicate with each other, through Raft system if needed, to maintain data consistency and handle server failures.

3.2 Communication

At this section we will describe the communication between the client and the key-value server, and the communication between the key-value server and the Raft server. Also we will describe the message types that are exchanged between the client, key-value server, and Raft server.

3.2.1 Message Types

The client, key-value server, and Raft server exchange messages in the form of JSON structures. Two classes were implemented to better decode, manage, and store the messages exchanged between the client, key-value server, and Raft server. These two classes are ServerJSON and RaftJSON. ServerJSON is a data structure used for communication between the client and the key-value server. It typically contains fields such as the command, and sender of the message. When the client sends a request to the key-value server, it encapsulates the necessary information in a ServerJSON message and sends it over the network.

Listing 1: ServerJSON

```
{
    "sender": "CLIENT | KV_SERVER",
    "commands": "example command"
}
```

The sender field can only have two values, CLIENT or KV_SERVER, which indicate whether the message was sent by the client or the key-value server, respectively. The commands field contains the command to be executed by the key-value server. The command can be one of the following:

- PUT: store a key-value pair in the key-value store.
- DELETE: delete a key-value pair from the key-value store.
- SEARCH: retrieve a key-value pair that match the specific pattern that was given to the system

On the other hand, RaftJSON is used for communication between the key-value server and the Raft server. It contains an extra field, which is a list of key-value server IDs, to indicate on which servers of the key-value store the data-import commands should be executed. Also the commands field is, unlike ServerJSON, an array of command strings.

Listing 2: RaftJSON

```
{
    "sender": "RAFT",
    "commands": ["example command 1",
                 "example command 2",
                 "example command 3"],
    "rep_ids": [1, 5, 6]
}
```

Both ServerJSON and RaftJSON messages are typically serialized into a suitable data format, such as JSON, before transmission over the network. This allows the messages to be easily parsed and reconstructed by the receiving party. The exchange of messages between the client, key-value server, and Raft server enables the coordination, synchronization, and fault tolerance mechanisms necessary for the reliable operation of the Key-Value Store application.

3.2.2 Communication Schema

The communication schema and path between the client, key-value server, and Raft server in the Key-Value Store application can be summarized in the following steps:

- Client to Key-Value Server: The client communicates with the Key-Value Server by sending requests over a network connection. When a client wants to store or retrieve a key-value pair, it sends a request to the Key-Value Server. The request is a ServerJSON object. The Key-Value Server receives the request and processes it by executing the appropriate action based on sender and the command type (PUT, DELETE, SEARCH). If the command type is PUT or DELETE, that is, some change should be made to the data stored on the servers, the Key-Value server reformat the message and send to Raft. Once the action is completed, the Key-Value Server sends a response back to the client.

- Key-Value Server to Raft Server: The Key-Value Server communicates with the Raft Server for maintaining consistency and fault tolerance. When a Key-Value Server receives a client request that requires updates to the key-value data, it interacts with the Raft Server to ensure that the updates are replicated across the cluster of Key-Value Servers. It sends append entries requests to the Raft Server, which includes the updates to the key-value data. The payload of the API call is a RaftServer object. The Raft Server replicates these updates to all servers and send the payload to all Key-Value Servers.

- Raft Server to Key-Value Server: The Raft Server communicates with the Key-Value Servers to maintain coordination and provide fault tolerance. The Raft Server receive the payload from the append entries API call and replicate it to all Raft nodes. It then send the message to all Key-Value Servers to process it. The payload of the API call is a string of RaftJSON object as said previously.

Overall, the communication between the client, key-value server, and Raft server in the Key-Value Store application follows a distributed architecture where the client interacts directly with the key-value server for data operations, while the key-value server interacts with the Raft server for consistency and fault tolerance. The Raft server coordinates the replication of updates and ensures that the key-value servers are in sync. This communication schema enables a reliable and scalable distributed key-value storage system.

4 RAFT consensus algorithm

5 CLI