Project Proposal: Distributed Sharded Key-Value Store

Group 41: Ruichen Zhang, Roger Wu, Kiran Pyles

April 5, 2025

1. Project Overview

The objective of this project is to design and implement a distributed key-value store that leverages sharding to achieve scalability, high performance, and fault tolerance. The system will store data as key-value pairs, distributing the dataset across multiple nodes (shards) using techniques such as consistent hashing. By integrating core distributed system concepts, the project aims to provide a practical demonstration of horizontal scaling, load balancing, and resilience against node failures.

2. Goal and Scope

Goal

Develop a fully functional distributed key-value store capable of handling high throughput for read and write operations, while dynamically managing data distribution as nodes are added or removed.

Scope

- **Data Distribution:** Implement sharding using consistent hashing to evenly distribute keys.
- **Fault Tolerance:** Ensure system resilience by replicating data across shards.
- **Scalability:** Demonstrate horizontal scaling by adding nodes with minimal data rebalancing.
- **Performance:** Optimize for low latency in key lookup and update operations.
- **Consistency:** Evaluate tradeoffs between eventual consistency and stronger consistency models under various network conditions.

3. System Architecture

The architecture consists of several key components:

1. **Client Interface:** Clients interact with the system using a simple API to perform put, get, and delete operations on keys.
2. **Coordinator/Router:** A routing layer maps incoming keys to the appropriate shard using a consistent hashing algorithm. This layer redirects client requests to the correct nodes.
3. **Shard Nodes:** Each shard is a server instance responsible for managing a subset of key-value pairs. Nodes communicate to replicate data for fault tolerance.
4. **Replication Mechanism:** Data will be replicated across multiple nodes. A replication strategy will ensure data availability in case of node failure.
5. **Rebalancing Service:** A background process will manage data redistribution when nodes are added or removed, minimizing data movement and system downtime.

4. Implementation Plan

1. **Design Phase:**
    - Define API specifications for client-server interactions.
    - Design the consistent hashing mechanism and replication strategy.
    - Create system architecture diagrams and flowcharts for key operations.
2. **Development Phase:**
    - Implement the client interface and coordinator/router.
    - Develop shard node functionality for storage, replication, and recovery.
    - Integrate the rebalancing service to support dynamic cluster changes.
    - Write unit and integration tests to ensure reliability.
3. **Testing Phase:**
    - Simulate network conditions such as node failures and high load.
    - Benchmark performance for read/write operations and rebalancing efficiency.
    - Adjust the replication and consistency settings based on test outcomes.
4. **Documentation & Deployment:**
    - Provide comprehensive documentation covering system design, API usage, and deployment instructions.
    - Deploy a demo instance for live testing and presentation.

5. Potential Challenges and Solutions

- **Data Rebalancing Overhead:**
  Challenge: Adding or removing nodes may trigger significant data movement.
  Solution: Implement incremental rebalancing using virtual nodes to limit disruption.
- **Network Partitions:**
  Challenge: Ensuring consistency during network splits.
  Solution: Use configurable consistency models (e.g., eventual vs. strong consistency) and implement quorum-based reads/writes.
- **Replication Lag:**
  Challenge: Delay in propagating changes across nodes could impact read consistency.
  Solution: Optimize replication protocols and monitor lag, integrating conflict resolution mechanisms if necessary.

6. Final Deliverables

- **Source Code:** Fully documented codebase for the distributed key-value store.
- **Design Documentation:** Detailed system architecture, API specifications, and design rationale.
- **Test Suite:** Automated tests covering unit, integration, and performance scenarios.
- **Demo Deployment:** A live instance or simulation demonstrating system scalability, performance, and fault tolerance.
- **Final Report/Presentation:** A comprehensive report summarizing design decisions, evaluation results, encountered challenges, and future improvements.