Project Proposal: PSDB - Python Sharded Database

Team Number: 31
Purva Dobariya (2024202001)
Malay Damani (2024201081)

Project Description

PSDB demonstrates a distributed Non-relational database by using Python and gRPC (for communication between nodes). This project is built from scratch which is inspired by modern document-oriented databases. This project handles basic CRUD operations which is doing sharding for horizontal scaling and provides high availability through replication. PSDB acts as a backend-only system with a simple CLI for user interaction and testing purposes. By building this project , we explore and implement distributed database principles within the timeline and also leveraging python's accessibility and gRPC's RPC framework.

Problem Description

Scalable and fault-tolerant databases are needed to handle big-data demands in modern applications which ensure uninterrupted service. Currently available relational-databases can be complex, making it valuable to create a simple , open-source system for our requirement. PSDB focuses on the challenge of designing a distributed database via sharding and availability by using python and gRPC (for communication).

Objectives

- Create relational-database with key-value pair and document-style storage.
- Implement sharding to distribute data across multiple nodes.
- Guaranteed high availability by using replication and failover mechanisms.
- Use Python for core logic and gRPC for inter-node communication.
- Implement scale testing to evaluate system performance under increasing load.
- Implement error handling and logging for easiness of debugging and maintainability.
- Demo the working prototype and create a detailed brief report of it

Features

1. Core Database Functionality
    - Provides CRUD operations (Create, Read, Update, Delete).
    - Store data persistently in document-style storage (eg: mongodb).
    - Simple, easy to use and intuitive CLI, for better user-interaction with the database (e.g., `insert`, `find`, `update`, `delete`).
2. Sharding
    - Partition data across multiple nodes using a shard key (e.g., hash-based partitioning).
    - Implement basic shard management to redistribute data so that nodes can be added or removed.
    - Execute routing logic to direct queries. (for appropriate shard)
3. High Availability
    - Use data replication across a primary-secondary model (at least 2 replicas per shard).
    - If the primary node fails, the secondary node must automatically handle failovers.
    - Implement eventual consistency via writes across all replicas.
4. Distributed Architecture
    - Implement gRPC-based communication between nodes (For replication, query routing, etc.).
    - Configuration file for defining cluster nodes and their roles (e.g., shard, replica).
5. Persistence
    - Use basic on-disk storage using files (e.g., JSON files per shard) for data persistence.
    - Implement a recovery mechanism to recover data after a node restart.
6. Scale testing
    - Evaluate system performance under increasing workload.
    - Test horizontal scalability by adding more nodes and measuring response time.
    - Ensure system stability under peak loads.
7. Error Handling & Logging
    - Implement logging for debugging and monitoring of load, latency and nodes.
    - Ensure graceful error handling for failures in sharding, replication, and communication.

Technical Stack

- Language: Python 3.11 (for rapid development and simplicity).
- Communication: gRPC (for efficient, cross-platform RPC).
- Storage: File-based persistence (JSON files for simplicity).
- Libraries to be used: `grpcio`, `protobuf` (for gRPC), `hashlib` (for sharding/security), `json` (for data storage).

Project Scope

- Core CRUD operations
- Sharding
- Replication
- CLI for testing
- Persistence
- Scale testing
- Error handling and logging

Timeline (1 Month, 2 Team Members)

Week	Tasks	Outcomes
Week 1	- Project setup (environment, gRPC boilerplate). - Design data model and sharding logic. - Define protobuf messages for gRPC.	- Initial codebase with gRPC setup. - Architecture design doc.
Week 2	- Implement CRUD operations. - Integrate a sharding mechanism (hash-based). - Develop CLI for testing CRUD.	- Single-node database with CLI. - Sharding prototype.

Week 3	- Implement replication and failover logic. - Build gRPC for inter-node sync. - Implement persistence (file-based). - Add logging for monitoring. - Start performance testing with load simulations.	- Multi-node cluster with replication. -Functional persistence. -Basic logging implementation.
Week 4	- Testing and bug fixing. - Optimize sharding and replication. - Conduct scale testing under various loads to evaluate performance and stability. - Finalize docs and demo prep.	- Working prototype. - Test cases and README. - Final Report - Scale testing results. - Documentation - Project Demo