Design Document - Part B: Networking Infrastructure for BLINK DB

Kumar Shresth 24CS60R51

March 31, 2025

1 Overview

This document details the design and implementation of the networking layer for BLINK DB. The goal was to build a high-performance, scalable server capable of handling multiple concurrent client requests efficiently.

Key features include:

- A TCP server using `epoll()` for efficient event-driven I/O.
- Support for the Redis Serialization Protocol (RESP-2) to handle client commands.
- Asynchronous request processing for high concurrency.

2 Architecture

The system follows a **client-server architecture**, where:

- The server listens on a port and processes multiple client connections asynchronously.
- Clients send **RESP-2 encoded** commands (`SET`, `GET`, `DEL`).
- The server decodes commands, interacts with the storage engine, and responds in RESP-2 format.

3 Key Components

3.1 Asynchronous TCP Server with `epoll()`

- Implemented a **non-blocking** TCP server.
- Used `epoll()` to efficiently manage multiple client connections.
- New connections and data readiness events are handled asynchronously.

3.2 Client-Server Communication using RESP-2

- Implemented a RESP-2 parser to decode client requests.
- Commands are extracted, validated, and executed by the storage engine.
- Responses are formatted in RESP-2 and sent back.

3.3 Handling Multiple Clients Concurrently

- When a client connects, its socket is added to the `epoll` instance.
- The server continuously listens for new requests and processes them efficiently.
- If a client disconnects, resources are cleaned up.

4 Implementation Details

4.1 RESP-2 Parsing

- Implemented a function to parse RESP-2 encoded messages.
- Ignored protocol markers (`*`, `$`) and extracted actual command values.

4.2 Handling Commands

- **SET key value**: Stores a key-value pair in the storage engine.
- **GET key**: Retrieves the value for a given key.
- **DEL key**: Deletes a key from the database.

4.3 Efficient Event Handling with `epoll()`

- Used `epoll_create1()` to initialize an epoll instance.
- Managed multiple client connections with `epoll_wait()`.
- When a client sends data, it is processed without blocking other connections.

5 Conclusion

The implementation ensures **efficient request handling**, **low latency**, and **high scalability**. The use of `epoll()` and RESP-2 parsing makes BLINK DB capable of handling multiple concurrent clients efficiently.