REPORT OF DECS PROJECT PHASE-I

M.Harshith-25m0834

1. GitHub Repository: https://github.com/harshith2302/KV-Server

2. Brief Description:

The Key-Value (KV) Store implemented as a C++ client-server application. supports **Create, Read, and Delete (CRUD)** operations over **HTTP**, using:

- **PostgreSQL** as the persistent database
- **In-memory LRU cache** for fast access
- **cpp-httplib** library for HTTP communication

The **Server** is a multithreaded C++ application that listens for HTTP requests. It manages data persistence by storing key-value pairs in a **PostgreSQL** database. To accelerate read operations, it uses an in-memory **LRU (Least Recently Used) Cache**. All database connections are managed by a custom **Connection Pool** to minimize latency.

The **Client** is a simple C++ command-line tool that provides a menu to interact with the server's API, allowing users to create, read, and delete key-value pairs.

3. System Architecture:

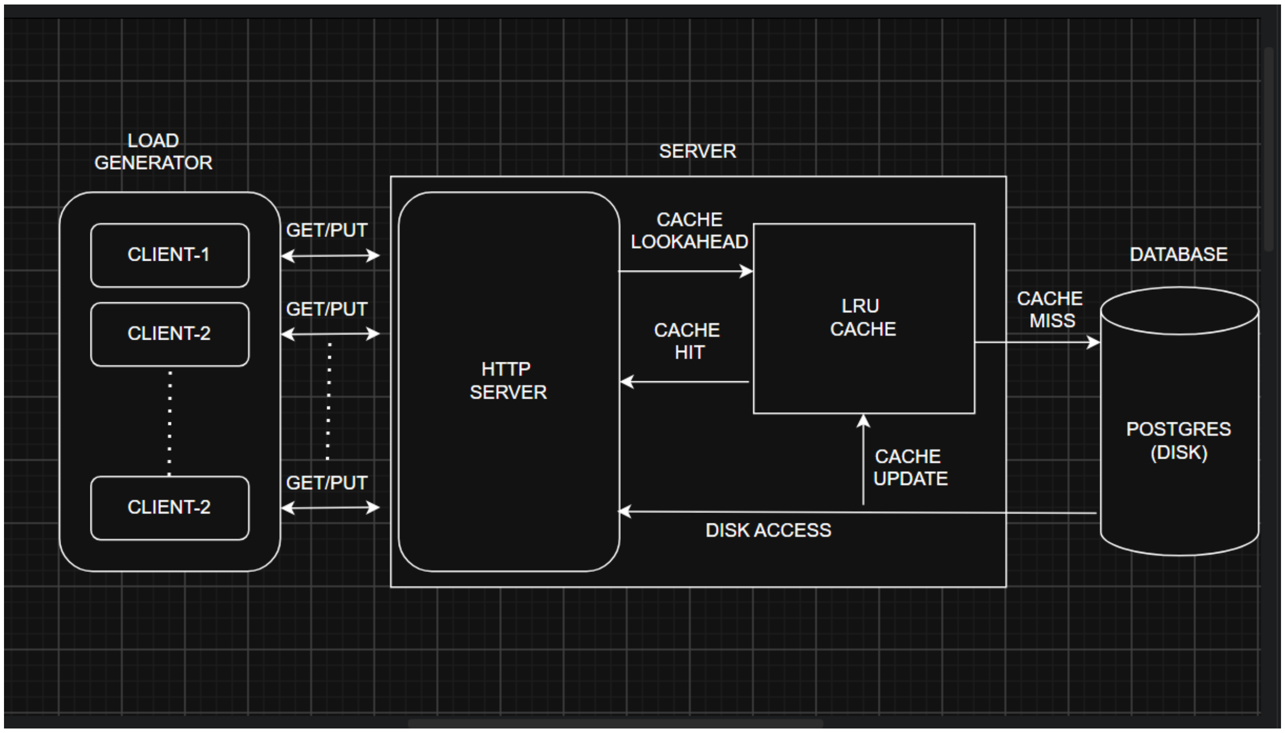

Fig : Overall flow of my system architecture

4. Explanation of Operations:

4.1 Read Request (/read):

1. The server first checks the LRUCache for the int key.
2. **Cache Hit:** If the key is found, the value is returned immediately. This is very fast.
3. **Cache Miss:** If the key is *not* in the cache, the server gets a connection from the DBPool, queries the PostgreSQL database, and (if found) stores the result in the cache before returning it to the client.

4.2 Write Request (/create):

1. The server gets a connection from the DBPool.
2. It executes an INSERT ... ON CONFLICT DO UPDATE command, which writes the data to the PostgreSQL database.
3. If the database write is successful, the server then **updates the LRUCache** with the new value. This ensures the cache stays consistent with the database.

4.3 Delete Request (/delete):

1. The server gets a connection from the DBPool.
2. It executes a DELETE command in the PostgreSQL database.
3. If the database delete is successful, the server **removes the key from the LRUCache**. This is crucial for preventing clients from reading "stale" data that no longer exists.