System Description and Architecture for phase 1

System Overview

The system is a **Key-Value (KV) Store** implemented as a multi-threaded **HTTP Server** with a caching layer and a persistent database backend. It processes simple HTTP requests (PUT, GET, DELETE) to store, retrieve, and remove key-value pairs.

Key Components

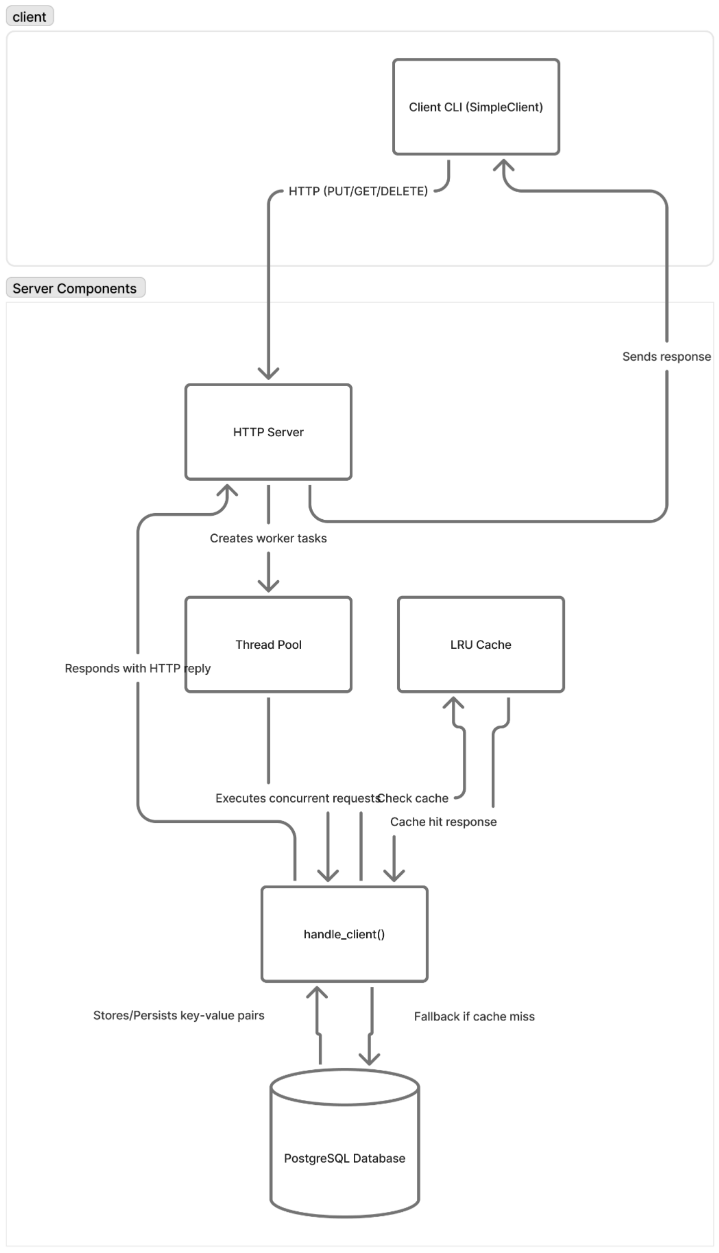
- `SimpleClient` **(Client Code):** A basic C++ class using **POSIX sockets** to connect to the server. It crafts and sends raw HTTP/1.1 requests (PUT, GET, DELETE) to the endpoint.
- `HTTPServer` **(Server Core):** The main component that sets up the listening socket and manages the application's lifecycle. It uses a **Thread Pool** to handle incoming client connections concurrently.
- `ThreadPool` **(Concurrency):** Manages a fixed number of worker threads. It uses a **task queue** and **condition variables** to distribute client handling tasks efficiently among its threads, ensuring high concurrency.
- `LRUCache` **(In-Memory Cache):** A thread-safe, memory-backed cache that implements the **Least Recently Used (LRU)** eviction policy. It is used to store frequently accessed data for fast retrieval (cache hits) and reduce database load.
- `Database` **(Persistence Layer):** A thread-safe wrapper around the `libpq` library (PostgreSQL client). It handles persistence by storing the key-value pairs in a PostgreSQL database table, ensuring data durability.

Request Flow and Data Management

1. A **Client** sends an HTTP request (e.g., PUT /kv/key, GET /kv/key) to the **HTTPServer**.
2. The **HTTPServer's** `accept_loop` accepts the connection and **enqueues** the client handling task to the **Thread Pool**.
3. A **Worker Thread** in the pool reads and parses the HTTP request.
    - **PUT Request:** The value is immediately written to both the `LRUCache` and the `Database` (Write-Through strategy).
    - **GET Request:** The system first attempts to retrieve the value from the `LRUCache` (**Cache Hit**). If not found (**Cache Miss**), it fetches the value from the `Database`. If the database returns a value (**DB Hit**), the key-value pair is written back to the `LRUCache` (Read-Through strategy) before being returned to the client.
    - **DELETE Request:** The key is removed from both the `LRUCache` and the `Database` .

4. The Worker Thread sends the appropriate HTTP response back to the client.

System Architecture Figure



🔗 Public GitHub Repository Link

GitHub Link: `https://github.com/Pankuu21/kvserver_25m0782`