Name: Patrick Johnson
Student ID: 5828453
Email: joh21451@umn.edu

Cloud Computing (CSCI 5980)

Coding Assignment 1: Designing a Simple Single-Server Key-Value Store

Project Repository: `https://github.com/overdodactyl/key_value_server`

1 Design Decisions and Justification

Flask was chosen as the web framework for this project due to its simplicity and ease of use for building RESTful APIs. Flask provides the necessary tools to quickly implement the key-value store server. Moreover, it has features that make it easy to quickly scale and run in different environments. In order to implement data persistence, a JSON file (the location defined by `DATA_DIRECTORY` and `DATA_FILE_PATH`) is occasionally saved to disk. The interval is defined by `PERSISTENCE_INTERVAL_SECONDS`. Shorter intervals will increase overhead, but also increase reliability. This allows the server to retain data between server restarts. On server start-up, the data file is looked for. If it exists, it loads the key-value store. If not, an empty one is initialized. In order to handle concurrent PUT and DEL operations on the same key, a thread lock (`key_value_store_lock`) was utilized. This ensures thread safety and prevents data corruption during concurrent access. The project also includes a logging mechanism to record all operations (GET, PUT, DEL) with timestamps. This design decision aids in debugging and auditing server activities.

2 Challenges

Managing concurrent access to the key-value store was challenging. To overcome this, a lock mechanism was used to ensure only one thread can modify the store at a time, preventing data inconsistencies. Implementing a data-saving mechanism that periodically writes data to disk required careful threading to avoid data corruption. This was achieved using a separate thread dedicated to this task.

3 Assumptions

One of the biggest assumptions made during this implementation is that a key can contain multiple values. In other words, when there is a `PUT` request on an existing key, it is appended to the existing values. A `DEL` request deletes all values for a given key. This implementation has several benefits but is not always used.

4 Improvements

This implementation could be improved in multiple ways, including but not limited to:

1. Database Integration: A database could be employed to improve the scalability of the server
2. Authentication: The current implementation allows anyone with access to the server to interact with it. Ensuring only authorized users can do so would improve security
3. Monitoring: It would be helpful to include some monitoring and metric performance to measure performance, usage, and downtime.
4. Additional methods: Other HTTP methods could be supported (e.g., PATCH)