Overview:

The design incorporates a primary-backup protocol for state machine replication, ensuring consistency across multiple nodes. The client-side components, including main functions, sockets, stubs, threads, and timers, facilitate robust and efficient communication with the server. These elements are designed to handle asynchronous communication and manage timeouts, providing a resilient interface for client operations.

On the server side, the main functions, socket communication, stubs, and threading are tailored to efficiently process client requests, manage the key-value store, and ensure data replication across backup servers. The server's architecture supports handling concurrent client connections, request processing, and failure recovery mechanisms. The design includes a detailed implementation of message passing for synchronization and consistency, with a focus on minimizing downtime and data loss.

Also, the system's modular design allows for easy extension and maintenance, with clear separation between the network layer, application logic, and data storage.

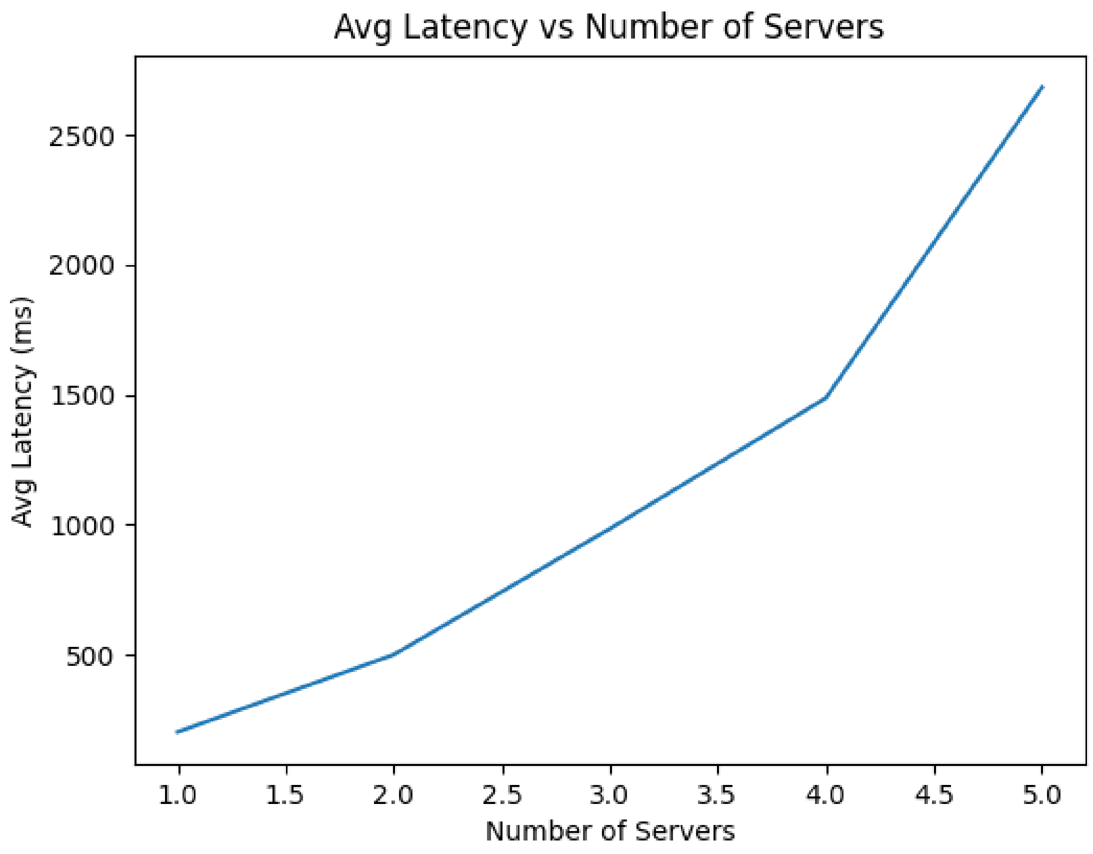
System Components:

1. Client Application Components: `ClientMain`, `ClientStub`, `ClientSocket`, `ClientThread`, and `ClientTimer`.
   - `ClientMain`: Initiates the client-side application, handling user inputs and interactions.
   - `ClientStub`: Facilitates communication with the server, encapsulating the network communication logic.
   - `ClientSocket`: Manages low-level network communication, including data transmission over the network.
   - `ClientThread` and `ClientTimer`: Support asynchronous operations and timing functions within the client application.

2. Server Application Components: `ServerMain`, `ServerStub`, `ServerSocket`, `ServerThread`, and key-value store logic.
   - `ServerMain`: Starts the server application, preparing it to process incoming client requests.
   - `ServerStub`: Processes client requests, performs operations on the key-value store, and prepares the responses.
   - `ServerSocket`: Handles accepting connections and communicating with clients.
   - `ServerThread`: Enables the server to process multiple client requests concurrently, enhancing scalability and efficiency.

3. Common Entities and Utilities: `Messages`, `Socket`, and `Makefile`.
   - `Messages` (`Messages.cpp` and `Messages.h`): Essential for serializing and deserializing communication between clients and servers.
   - `Socket` (`Socket.cpp` and `Socket.h`): Simplifies socket programming for network communication.
   - `Makefile`: Automates the compilation process, ensuring proper build with dependencies.

Test Result:



Latency increases proportionally with the addition of servers, which aligns with expectations. This is because incorporating more servers extends the duration required to distribute logs across all servers, thereby impacting overall latency.