Executive Summary

1. Assignment Overview -

The goal of this project is to design and build a distributed, multi-threaded key-value store (KV-store) system that uses the Paxos consensus algorithm to achieve fault tolerance and consistency among replicated servers. The major goal is to allow concurrent client operation while ensuring consistency during write and delete operations, even in the event of server failure. Building on the work done in Project #3, which replicated the KV-store over five servers using Two-Phase Commit (2PC), this assignment attempts to replace the 2PC protocol with the more durable Paxos protocol, guaranteeing that consensus is reached across all replica servers in the system.

The implementation will center on the three fundamental Paxos roles: proposers, acceptors, and learners, with the goal of reaching consensus on write operations (PUT and DELETE) even if some replicas fail or become unresponsive. Clients will be able to interact with the system via both interactive and scripted inputs, including basic operations like GET, PUT, and DELETE. A critical component of the project will be mimicking server failures by occasionally "failing" the Acceptors, pushing the system to recover while retaining consistency and fault tolerance.

This assignment also includes the deployment of the distributed KV-store system utilizing Docker containers, which ensures that the system is scalable and easily replicated between multiple nodes. The project simulates realistic distributed system circumstances by incorporating random failure mechanisms into the Paxos protocol, demonstrating Paxos' ability to maintain consistent operations in the face of failure.

Overall, this assignment seeks to reinforce ideas such as distributed systems, consensus techniques, fault tolerance, and containerization. By applying these concepts to a real-world, fault-tolerant KV-store system, students will get practical experience in developing resilient distributed applications and managing the complex nature of state synchronization across multiple replicas.

2. Technical Expression -

This task was both tough and insightful, especially when it came to developing the Paxos protocol and integrating it with a fault-tolerant key-value store system. The most difficult component was understanding and implementing the Paxos roles—Proposers, Acceptors, and Learners—as well as ensuring that the protocol achieved consensus on updates even when certain acceptor threads "failed" at random during execution. The Paxos algorithm is not straightforward; it necessitates careful thread synchronization and ensures that messages are successfully delivered between the proposers, acceptors, and learners. Implementing fault tolerance by randomly terminating and restarting acceptor threads increased complexity because it required me to handle state recovery and ensure that the system remained consistent even after failures.

Another problem was ensuring that the key-value store could manage several concurrent client operations without resulting in race situations or data inconsistency. This necessitated careful handling of thread synchronization and client requests, as well as the creation of a system capable of supporting concurrent client interactions while maintaining the shared key-value store's integrity.

Docker provided a straightforward and efficient approach for replicating communication between server replicas and clients, and it was satisfying to watch the system scale as additional client containers interacted with the server. Overall, the project considerably increased my understanding of distributed systems, fault tolerance, and consensus methods, and it was exciting to see a system that could withstand errors while still providing consistent operations.