Design Document
Distributed Key Value Store

In this project, we have created a Rest API service using Jersey Framework in Java programming language. The operations supported are PUT, GET, DELETE. We have made the Rest API service distributed. We have opted for a CP system. 5 nodes were given for testing this project. We are using docker to run the project.

Primary Backup Replication protocol:

Replication protocol used for this project is primary backup protocol. So the ordering of operations is done by the primary. Each node stores the (key, value) pair in an in-memory Hashmap.

Primary Election:
Each node has a Zookeeper library with it as configuration manager. Zookeeper is also used for electing primary. Zookeeper is an AP system but if we use the sync method provided by its API, it will behave like a CP system. So we used the sync operation before getting the leader.
The algorithm used is quite simple and popular. This is not our idea. Each node will create a sequential and ephemeral Znode in a given directory. (Znode is a directory in zookeeper terminology). Here sequential means zookeeper will numebr the znode created. Ephemeral implies that when a node gets disconnected from majority group, the znode created by that node is deleted. Leader is the node which has created a directory with smallest sequence number. So whenever a majortiy of replicas are connected, a leader is chosen among them.

Handling READ request:
When a client sends a read request to a server, there are two cases.
1.If the accessed server is primary, it will send back the reply right away.
2. If the accessed server is backup, the request will be directed to primary.

Handling PUT/DELETE request:
When a client sends a PUT/DELETE request to a server, there are two cases.
1. If the accessed server is primary, it will replicate it to all the backups and also store in its own hashmap.
2. If the accessed server is backup, the request will be directed to primary.
3. Each replica will also store the operation in a Log. Each item contains the operation type whether “put” or “delete” , key , value (in case of put).

Handling failures:
1. When a node is crashed, the remaining replicas will continue to serve requests, if they have the majority, implying a leader is elected. Otherwise, if no majority is there, implies no leader, then the system will be unavailable.

2. If a majority of replicas is formed, then if any stale nodes are present in the majority, they will get updated in the group from the node whichever has up-to-date information. Then a leader is elected by following the process described above.

3. Whenever a new node is added into the active group, that node will be brought up-to-date too.

4. When updating more than one node, each node will be passed only those operation which it is missing. So, network is not used unnecessarily.

Expected behavior:

1. The system will perform as a single node without any errors when no failures are present.

2. When a node is failed, the system will try to reach it with a timeout value of 2 seconds. If the timeout is finished, then that node is considered not reachable.

3. A request sent to a node, which cannot reach a leader will not respond to the request until it reconnects to the leader. The client will not get any response in this time period.

4.When a majority is formed and if some nodes have stale information, then the system won't be available until all the replicas are brought up-to-date.

5. The timeout set to reach a node is set as 2 seconds.

6. The system will be consistent at any cost.

7. The system will perform without any bugs or inconsistencies, even if a majority is not present for long time periods.

Testing the system:

1. The system has passed all the test cases given by the TA, Nikhil. However, th give test suite doesn't contain any node failures.

2. To emulate network latency, we have used docker pause functionality. A paused docker container will not respond to any requests. We can unpause the docker again by using “docker unpause <contianerID>” command. So we have induced different latencies and different partitions. The system has performed well and reached all of its expectations.

3. To test for crashes , we have used “docker stop <containerID> command. We have induced different crashes and the system performed without any bugs and did not violate any expectations. Our system also performed correctly even if the crashed containers are restarted again by using “docker restart <containerID>” command.