Assignment 4 - Design Document

De Li - u4o8 Hanson Chun - f5l8

Kv node is abbreviated to node. R is the replication factor. X is the number of active nodes.

Nodes connect to the frontend using TCP. A buffer I/O is used to read and write over the TCP connection. Each line sent over the connection is a single message with the format "<length of message> <message>". When first connected, the node will initially send its id as the first line. The frontend sends requests (e.g. GET) in JSON to the node and the node responds with the result in JSON. It is assumed that the node has failed if an error occurred during TCP connection. Any time a node fails, the frontend will mark any outgoing requests for that node as unavailable.

We keep track of which keys are replicated on which nodes and whether the key has been marked as available or not. In addition, we keep track of the version of the key (starts at 0 and increments for each PUT/TESTSET call) and the number of ongoing RPC calls for each key.

RPC calls are serialized in a queue and processed one by one by a MAIN thread. For each call, MAIN sends the corresponding requests to each of the nodes. Since the RPC calls are serialized and TCP guarantees in order transmission, each key will have the same value for each replica at the time of each request. The first response received from the nodes is sent back to the client (since any subsequent response will be the same). If all responses are unavailable, "unavailable" is sent to the client and the key is marked as unavailable (we will not retry the request to preserve the order of RPC calls). If the key is new, random nodes will be selected as the nodes for the above steps (i.e. this is initial replication of a key).

Replication is done concurrently in separate goroutines. Each time a node joins or fails, replication routines are launched to replicate a set of keys to a target node. When a node joins, the set of all keys is replicated to the joining node if X < R. When a node fails, for each key the failing node had stored, select a node that is not the key's replica and set it as the target node for the key. Then the frontend replicates each set of key to the associated target node. When the target node fails, we re-partition any keys that have not been replicated to the active nodes and proceed like above. This maintains min(X, R) replicas when all replication succeeds.

The replication routine gets the values for each key from other replicas. When it receives a value for all the keys or when all replicas respond, it PUTs the values into the target node. Then we check for keys that are successfully replicated. A key is considered successfully replicated by the front end if the target node has the latest version of the key, if there are still replicas for the key*, if there are presently no RPC calls**, and if the key has not been marked unavailable. A key is dropped from replication if there's currently no replicas with the key value. This repeats until all keys are replicated.

* A replica can fail, the replication process put the key to target node, then remaining replicas fail. However, the key can't be replicated successfully because e.g. CMD get-replicas-of k can return 0, then the next call returns 1
** (very unlikely scenario) RPC 1 using failing nodes and returns unavailable, then RPC 2 using failing nodes + new replica, and new replica returns replica value to RPC 2 before RPC 1