EVENTUALLY CONSISTENT KEY VALUE STORE

MOTIVATION:

Designing and building an eventually consistent key value store with high availability by using principles of the Dynamo and analyzing the impact on key value store in terms of consistency, latency and request failures.

ARCHITECTURE

The key value store follows leaderless architecture. The key features of the architecture are:
Replication: Replication of data will be done asynchronously to achieve high availability. Each write will go to a random node. This node will be responsible for replication data to the rest of the nodes asynchronously. This means the write request will be returned to the caller before it is replicated to all the nodes. The write conflicts will be resolved using Data versioning with Vector clocks (Fig2).
SWIM Membership and Failure Detection: Failure detection component: detects failures of members. Dissemination component: the dissemination of information about the node that have recently joined or failed (Fig1).
Anti-entropy: Built **chain** using hash of key,value and successor node hashkey. If the head of the chain is equal between any two nodes then the key value stores are consistent with each other. Best case 1 comparison and worst case n comparisons (Fig3).
Response to client: For set request, when replica receives response from W replicas, client receives a success response.
For get request, when replica receives response from R replicas, conflict resolution between values is done. Most of the time for any data, the new version will subsume the old version but in case of concurrent requests, all available versions of the data will be returned.
Quorum configuration of R+W<N is maintained to achieve eventual consistency.

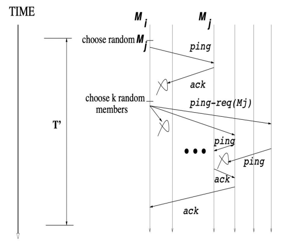


Fig.1

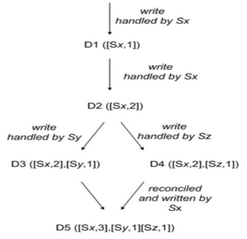
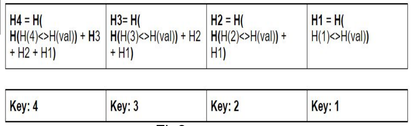


Fig2.



H4 = H(H(H(4)<>H(val)) + H3 + H2 + H1)	H3= H(H(H(3)<>H(val)) + H2 + H1)	H2 = H(H(H(2)<>H(val)) + H1)	H1 = H(H(1)<>H(val))
Key: 4	Key: 3	Key: 2	Key: 1

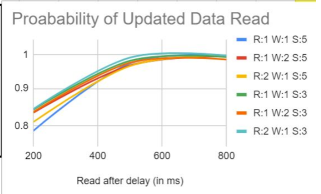
Fig3

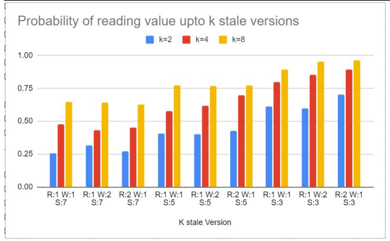
CHALLENGES HANDLED:

1. Key value store handles concurrent requests from multiple clients
2. Client can send another request to system before getting response for the previous request without overwriting previous data

FINDINGS and CONCLUSIONS:

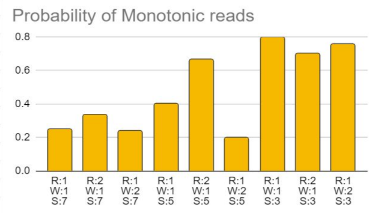
1. The chart (on right) shows that the KV store do eventually become consistent. As delay between write and read increases, the read values obtained are updated and consistent between different replicas.





2. The chart (on left) shows that under conditions where N >> R and N >> W, probability of getting updated read values immediately after multiple writes is very less. We show this with different degree of staleness and compare with different server configurations.

3. From the third chart (on right), we see that the monotonic read consistency increases with decrease in replica size with fixed quorum configuration. With the same server size, R>W has more monotonic read consistency.



** The experimental setup incorporated delays in the network.

4. Merkle synchronization fails frequently during write intensive operations (due to keys updating frequently). Therefore merkle timeouts should be more for write intensive key value stores.

CONCLUSION: From all the results, we observed that having less write latency and more read latency results in consistent data. As eventually consistent stores cannot promise strong consistency, a good staleness bound with respect to time and versions and appropriate R and W to reduce latency is our choice of building a kv store.

FUTURE WORK:
1. Client Side Resolution in case of multiple read value response.
2.change of dissemination component of gossip protocol to infection style to reduce multicasting
3.Add suspicion mechanism to failure detection to reduce false positives.