Dynamo Based Distributed Key-Value Store

Anand Kumar, Ajeeta Asthana
New York University

Abstract

DynamoDB is a highly available, fully managed, key-value NoSQL database. It is known for its 'always-on' experience and single-digit millisecond performance at any scale. Based on [1], we examine CAP theorem consequences of offering availability, handling network partitions with a weaker consistency model. We referenced [2] for detailed implementation.

We examine the effects of consistent hashing, sloppy and strict quorums in our design. Process failures are handled assuming a fail-stop model (a failed process never rejoins the cluster, therefore no need of hinted handoff mechanism). View reconciliation in the scenario of multiple node failures (Nodes/processes redistribute key ranges ownership to maintain N -> the replication factor) are tested as well.

Methods

We have implemented a distributed key-value data store based on Amazon's DynamoDB.

The following **operations** are supported :

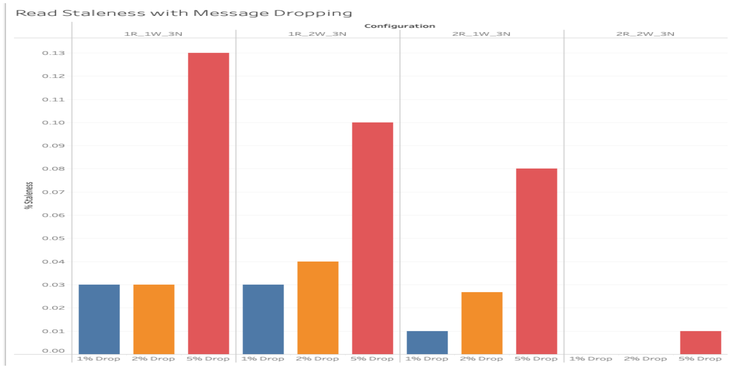
- **put(key, value, context)** : Puts the value corresponding to the key and its context. The context is always increasing.
- **get(key)** : Fetches the value, context corresponding to the key.

The following features have been implemented :

- **Partitioning** : The partitioning is done using consistent hashing. The keys are distributed evenly across all the nodes. Our setup supports keys in the range of $0 - 1000$ and it has 10 starting nodes.
- **Vector Clocks with Reconciliation during Reads :** We are using context while writing and returning the value with highest context for the read operation.
- **Support for Sloppy Quorum :** The system can be configured for R+W < N for ensuring high availability.
- **Gossip Protocol :** A gossip timer is implemented to propagate membership changes and maintain an eventual consistent view.

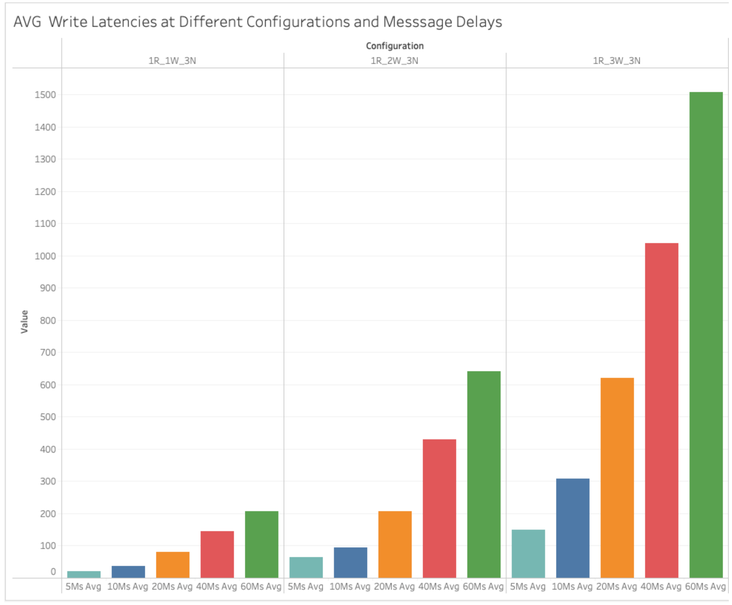
Measuring Read Staleness with Message Dropping

We measured the read staleness with 4 different configurations namely, 1R_1W, 1R_2W, 2R_1W and 2R_2W with varying message delays.

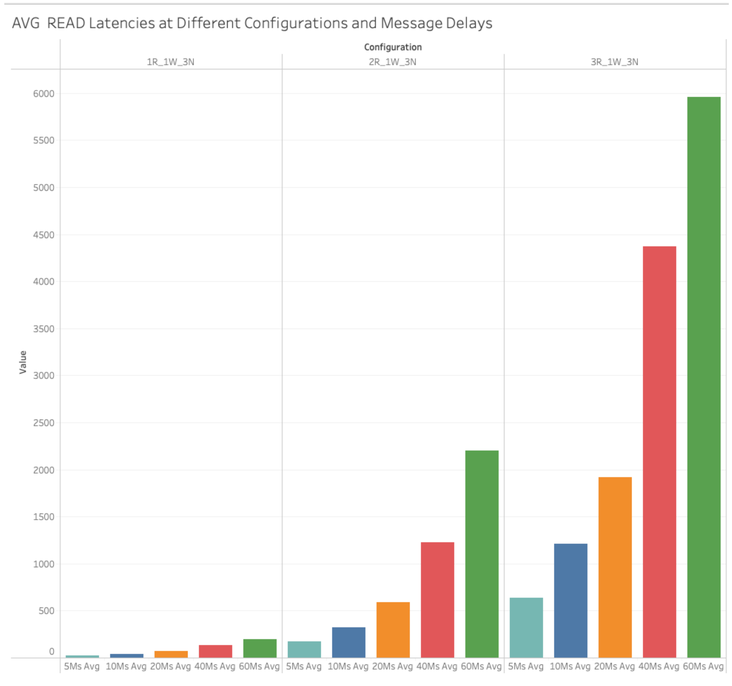


Measuring R/W Latencies With Message Delays

We measure mean latency for read and write operations over increasing message delay times. For Write Latencies we observed that, as we increased the parameter 'w' the write latencies increased.



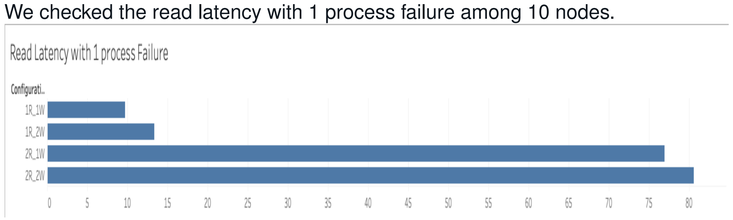
b). Avg Write Latency with varying msg delays



c). Avg Read latency with varying msg delays

Read Latency and Process Failure

We checked the read latency with 1 process failure among 10 nodes.



d). Avg Read latency with 1 process failure

Conclusion

As part of the project, we ran the following experiments.

Read staleness with message dropping: The staleness of the data for various R, W and N configurations with drop rates 1%, 2% and 5% was carried out. We observed that as we move from sloppy (R+W < N) to strict quorums (R+W > N) the consistency increases. This is because in strict quorums, the return value is chosen as the latest from the majority (R) of replicas.

Read/Write Latency with message delays: Time taken to complete the read/write operation was measured for different network latencies and various R, W , N configurations. We observed the average latency for both read and writes increases as we go from sloppy to strict quorums and increasing message delay rates from 5 ms to 60 ms.

Read Latency under a single process failure: After failing a single node and carrying out the read/ write operations, we observed that the average read latency increases from 9 ms at (1R,1W) configuration to around 80 ms at (2R,2W) configuration. This is because of increase in read quorums and the view reconciliation because of node failures.

References

[1] Giuseppe DeCandia et al. "Dynamo: Amazon's Highly Available Key-Value Store". In: *ACM, SOSP '07* 41.6 (), pp. 205–220.

[2] Jonathon Henderson. "Destructuring Dynamo, sep 2020". In: ().