Building a Practical Key-Value Store

Keyur Panchal Jayanth Reddy
kp3159 nr2686

1 Main Goal

The main goal of our project is to build a practical, scalable key-value store.

We plan to build a key-value store that can be used to power a production-level system. It will be designed to mitigate several assumptions that do not always hold at scale. Several optimizations will also be implemented to make it a viable system that can be used.

The motivation behind this project is described in section 2. Some of the related work we have looked at is in section 3. We talk more concretely about the design choices and optimizations that we plan to make in section 2. Finally, we describe some of the evaluation metrics that we plan to use in section 5.

2 Motivation

The primary motivation for our project is the 'Paxos Made Live' paper [1]. The idea is to take a consensus algorithm and build a production-ready system around it. This task is challenging since several assumptions are made in the realm of theoretical distributed systems that do not easily translate to real-world systems. Much needs to be built on top of the system in order to make it practical and safe to use. For example, Raft [10] and Paxos [8] assume that the replicated log stored at each node is durable. This is a fair assumption as the emphasis is on ensuring that consensus can be reached in a distributed setting. However, while storage systems are fairly fault-tolerant, it cannot be assumed that they never fail – this may lead to Byzantine failures, in which case neither Raft nor Paxos guarantee safety.

There are two main challenges with this project that interest us. The first is that the system must ensure that the assumptions made by the underlying consensus algorithms hold, and it should mitigate the effects when they do not. For example, we can mitigate the risk of assuming that storage is durable by adding redundancy at the storage level, or implementing data integrity checks (and perhaps even data self-correcting algorithms) to ensure that safety is never violated.

The second challenge of interest is performance. We understand that while correctness is of utmost importance, it is unlikely that a system is used if that correctness comes at the cost of poor performance. Hence, we also seek to implement several optimizations to ensure that the system performs well.

3 Related Work

As stated earlier, the primary motivation of this project was the 'Paxos Made Live' paper [1]. We shall implement several design optimizations, initially introduced by DynamoDB [2] [3] [12] and Cassandra [7], such as Read Repair, Hinted Handoff, and Anti-Entropy. We would also like to look into more recent systems such as Garnet [9] and ShadowFax [6] that have made interesting optimizations that may prove useful in the design of our system.

4 Design

While the complete design of the system is still in the initial stages, we are looking to build the key-value store in Golang, to leverage the high degree of performance that Go offers via goroutines. We will use an existing Raft library [4] to maintain the write-ahead log, to allow for linearizable reads and writes.

Some of the assumptions that we are looking to tackle, and the idea we have in mind to detect/fix them, include:

- Durable storage: We will implement redundant storage and data integrity checks, and leverage more resilient storage options such as RocksDB [5], FASTER [6], or some other custom file system.
- Reliable networks: We will use gRPC communication between nodes. We will try to ensure that RPCs are designed to be idempotent, so we can safely assume exactly-once or at-most-once semantics.

Some of the optimizations we're looking to implement are:

- Data partitioning: Partitioning data across nodes allows the system to scale out better, and balances the load across multiple nodes, which improves read performance. This will likely be implemented using consistent hashing.
- Concurrency: Go provides goroutines, a concurrency primitive built into the language itself, that we will leverage to improve performance.
- Anti-entropy/Gossip: This ensures that less data needs to be transferred during log replication.
- Read repair: Read repair allows nodes with stale data to receive the latest version of the data when a read is attempted.
- Hinted handoff: This is an optimization for writes. It allows nodes to store a copy of data that does not belong to them. They can later send it to the node that should own that data when it comes back up.
- Log Replicas: Certain nodes may be designated as log replicas: they only store the write-ahead log and do not store the actual key-value data. This optimization helps to replicate the log faster, and allows nodes with less compute capability to be used in the cluster of nodes.

Please note that the above features are not an exhaustive list. We are also looking into several other features. Moreover, given the short timeline of this project, we will likely limit the number of features implemented depending on the challenges faced during implementation.

5 Evaluation

Testing distributed systems is challenging. Our primary focus will remain on correctness, and we will evaluate our system under various failure scenarios to ensure it works correctly. We have looked into Toxiproxy [11], a tool to help simulate network failures. We will also evaluate our system's correctness in the face of storage failures.

Next, we will benchmark the system with and without each optimization. This will help us evaluate how much difference each optimization made. This is workload-dependent, and we will attempt to find or create a suitable workload to evaluate each optimization.

Finally, we may benchmark our system against another popular distributed key-value store. While the results may not be fruitful, it may be interesting to see how close our project can get.

References

[1] Tushar D. Chandra, Robert Griesemer, and Joshua Redstone. "Paxos made live: an engineering perspective". In: *Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing*. PODC '07. Portland, Oregon, USA: Association for Computing Machinery, 2007, pp. 398–407. ISBN: 9781595936165. DOI: `10.1145/1281100.1281103`. URL: `https://doi.org/10.1145/1281100.1281103`.

[2] Giuseppe DeCandia et al. "Dynamo: Amazon's highly available key-value store". In: *ACM Symposium on Operating System Principles*. 2007. URL: `https://www.amazon.science/publications/dynamo-amazons-highly-available-key-value-store`.

[3] Mostafa Elhemali et al. “Amazon DynamoDB: A scalable, predictably performant, and fully managed NoSQL database service”. In: *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. 2022, pp. 1037–1048.

[4] etcd-io. *Raft*. `https://github.com/etcd-io/raft`. 2024.

[5] Facebook. *'RocksDB'*. `https://research.facebook.com/file/247027027415678/RocksDB-Evolution-of-Development-Priorities-in-a-Key-value-Store-Serving-Large-scale-Applications.pdf`.

[6] Chinmay Kulkarni, Badrish Chandramouli, and Ryan Stutsman. “Achieving high throughput and elasticity in a larger-than-memory store”. In: *arXiv preprint arXiv:2006.03206* (2020).

[7] Avinash Lakshman and Prashant Malik. “Cassandra: a decentralized structured storage system”. In: *ACM SIGOPS operating systems review* 44.2 (2010), pp. 35–40.

[8] Leslie Lamport. “Paxos made simple”. In: *ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)* (2001), pp. 51–58.

[9] Microsoft. `https://www.microsoft.com/en-us/research/blog/introducing-garnet-an-open-source-next-generation-faster-cache-store-for-accelerating-applications-and-services/`. 2024.

[10] Diego Ongaro and John Ousterhout. “In search of an understandable consensus algorithm”. In: *2014 USENIX annual technical conference (USENIX ATC 14)*. 2014, pp. 305–319.

[11] Shopify. *Toxiproxy*. `https://github.com/Shopify/toxiproxy`. 2024.

[12] Swaminathan Sivasubramanian. “Amazon dynamoDB: a seamlessly scalable non-relational database service”. In: *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. SIGMOD '12. Scottsdale, Arizona, USA: Association for Computing Machinery, 2012, pp. 729–730. ISBN: 9781450312479. DOI: `10.1145/2213836.2213945`. URL: `https://doi.org/10.1145/2213836.2213945`.