SDLE T02G04

Gustavo Costa up202004187
João Oliveira up202004407
Ricardo Cavalheiro up202005103

November 7, 2023

1 Introduction

Combining the principles of local-first applications with the convenience of collaboration platforms like Google Docs allows for a rich user experience with few drawbacks.

This paper talks about the architecture and implementation details of a local-first, collaborative shopping list application.

2 Objective

The objective of this project is to explore the world of massive, decentralized, and local-first applications by developing an application capable of supporting millions of users, as well as understanding the principles and benefits of CRDTs, coupled with data sharding techniques, such as consistent hashing.

3 Non-functional requirements

The following non-functional requirements are a set of specifications meant to enhance the functionality of the system by describing its operating capabilities and limits.

- The application should allow multiple users to make changes to a shopping list at the same time and deal with possible conflicts that may arise. E.g data merging
- The application should work even if the user does not have an internet connection at a given moment. When the connection is resumed the data should be pushed to the backend in order for it to be synchronized among all collaborating users.
- Users should only have access to the shopping lists of which they know the ID.

4 Architecture overview

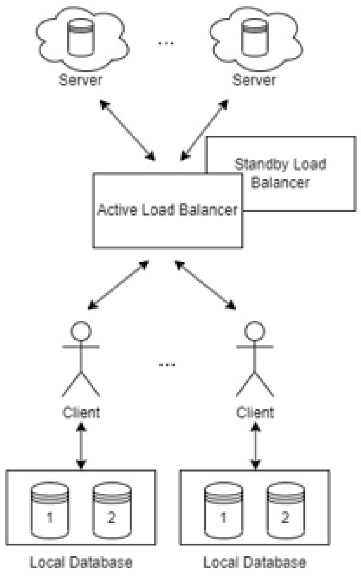

Figure 1: Architecture suggestion diagram

4.1 Local-First Design

Local first is a design principle that prioritizes the importance of local operations and data availability. By emphasizing local operations, distributed systems can strike a balance between responsiveness and resilience, ensuring a smoother and more efficient user interaction. In order to achieve this, we will use CRDTs together with Vector Clocks to represent the state of each shopping list. Combining both effortlessly facilitates data synchronization and versioning, particularly in situations where numerous users are simultaneously updating information.
Each client has two copies of the shopping lists he has ever accessed. One is for both read/write operations and the other for only reading. Whenever there's a write to the main one, it is propagated to the secondary one.

4.2 Load Balancing

Load balancing is crucial for optimizing system performance by evenly distributing incoming traffic across multiple servers. A load balancer is aware of the shopping list to server mapping and redistributes the requests. Using a dual-load balancer setup, one active and one on standby, enhances system reliability. The active balancer manages traffic, while the standby serves as a seamless backup, ensuring uninterrupted operation and mitigating the risk of downtime due to server overload or failure. The system routinely performs health checks on the load balancer, swapping it with the one on standby, if it is needed.

4.3 Consistent Hashing

Consistent hashing makes it possible to distribute data so that the addition or removal of nodes causes less amount of reorganization of the stored keys, which facilitates system scaling. The essential point is that the distribution system in question is not contingent on the quantity of servers. In order to guarantee that the load among the servers is evenly distributed, we will be using **virtual nodes**, which allow for a finer distribution of the hash space, as well as ensure data availability in the face of failures.

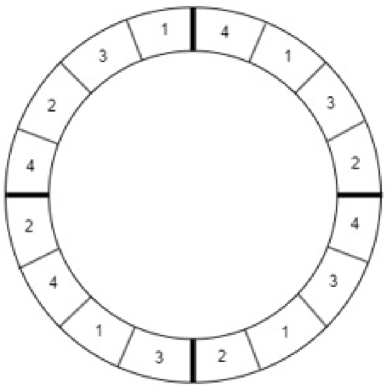

Figure 2: Hash Ring

4.4 Replication Approach

We'll explore two types of data replication: local replication and server replication. Following the principles of local-first applications, we thought that it was probably for the best that each user has a local copy of all the shopping lists he has ever accessed. This mechanism allows for failures that otherwise would need a network connection to retrieve any lost data. In order to ensure data availability on the cloud, we would also be introducing redundancy by replicating the data server side, propagating the writes to a node among the next clockwise N nodes on the hash ring.

4.5 Failure Handling

We'll start this subsection by prefacing that no system is truly failure-proof, especially when talking about large-scale distributed systems. With this in mind, there are multiple points of failure that we need to take into consideration:

- No network connection: The local-first ideals allow for offline work without compromising the user experience. Obviously, there can't be collaboration without network access, but each user can still work on his own version of the shopping list that is then synced once the network is back online.
- Load balancer issues: In case of issues with the active load balancer, the routine health checks should deploy the standby load balancer into action until the main one is back online. Obviously, we can't have infinite redundancy, so there's a limit to how many standby load balancers there can be.
- Gossip-based node health checks: Periodically, nodes should check the health of their neighbors, to ensure data availability and correctly distributed loads.
- Main servers go offline: In the event of failure of the main read/write servers, the load balancers should be capable of redirecting traffic to the backup servers. Also, with server failure in mind, a sloppy quorum system should be in place to ensure the latest data is transmitted, even if multiple servers are offline.

5 Conclusion

Our shopping list application's design is based on the Amazon Dynamo paper, which aims to provide a user-friendly local-first solution. We hope to ensure data availability and integrity while offering consumers a smooth experience, whether or not they have connectivity.