Shopping Lists on the Cloud

Design and Technology Choices

André Filipe Cardoso Barbosa - up202007398
Guilherme Cunha Seco Fernandes de Almeida - up202008866
Tiago Filipe Magalhães Barbosa - up202004926

Large Scale Distributed Systems - MEIC

2023/2024 - First semester

1 Introduction

This report outlines the design and technology choices for a local-first shopping list application that aims to provide a seamless user experience while guaranteeing high availability and data synchronisation. The application allows users to create and share shopping lists, with support for concurrent modifications and flexible conflict resolution.

The architecture features client-side applications with local database storage, partition-aware routers and a server ring inspired by Amazon Dynamo[1] to ensure data storage and replication.

The design complies with local-first principles, prioritises scalability for millions of users and provides a framework for future work.

The report breaks down the design decisions and their justifications, laying the groundwork for a scalable and responsive shopping list application.

2 Design Overview

In order to achieve the defined objectives, such as high availability, scalability and guaranteeing data consistency, our architecture takes a solid and effective approach to tackling these challenges.

In the following sections, we will detail each component of our architecture. We will commence with the client-side component, proceed to the intermediate routers, and ultimately delve into the cloud-based server ring. Furthermore, we will examine overarching decisions encompassing conflict resolution strategies.

The starting point for our architecture is the "local-first"[2] principle, which serves as the basis for the user experience. In the cloud layer, inspired by Amazon Dynamo, we adopted the approach of a ring of servers with data partitioning and replication. This choice aims to provide high availability without compromising data consistency. We used partition-aware routers to distribute the data evenly between the servers, avoiding bottlenecks.

We also use Conflict-Free Replicated Data Types (CRDTs)[3] and Vector Clocks[4] to deal with possible conflicts. The selection of technologies considers high availability and scalability, ensuring that our application can be easily scaled to serve millions of users.

Below is an overview of our architecture:

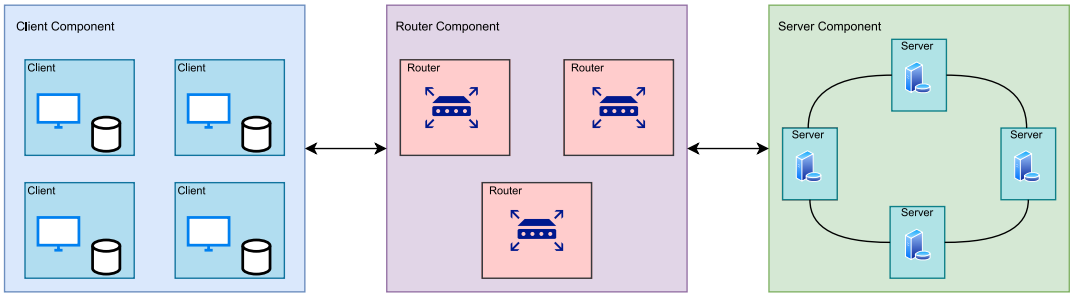

Figure 1: Architecture overview

3 Client-Side Application

The local-first principle is the basis of our client-side design. This ensures that the data belongs to the client, allowing them to access and modify their shopping lists even when they are offline, not requiring constant connection to the cloud for proper operation. To achieve this we will use a small and fast local database and will then send the changes to the servers so other clients are aware of these modifications.

With this in mind we will design a user-friendly interface to allow our users to create new shopping lists as well as share them with others using its unique ID. Users who have this ID will be able to collaborate on the shopping lists simultaneously. We will allow basic operations such as adding products, updating products in the list, marking items as purchased and defining quantities for each product.

4 Router Component

We will use an intermediate component in our architecture between the client and the cloud-based server ring to allow us to handle client requests, decide which server to send the request to and then guarantee the client that their changes have been acknowledged by the cloud.

This component will consist of routers that all clients will know and that, after receiving a request from a client, will calculate which partition the item's key belongs to, using consistent hashing, and then see who is the server node coordinator of that partition. Then it will send the request to this server to store the new changes of the shopping list.

Due to the very nature of the distribution of partitions across the ring, there will be a natural load balancing on the servers, thus avoiding overload on any of the servers in the ring.

We will use 3 routers to allow that even if one of them fails there are others for clients to connect to. But for this we have to assume that the routers will be simple systems and that they will be available 99.99% of the time, meaning that the probability of the 3 routers failing at the same time will be super low, leading us to assume that we will not have any bottleneck or failure in the system due to this component.

The routers, as mentioned above, will be aware of the partitions of the server ring, as this assures that all operations in the cloud are much faster than sending the request to a random server and then trying to find out who is the coordinator of that partition.

As the user base and data load grow, the router component must be capable of scaling to handle increased client requests and routing tasks. Our architecture is designed to support horizontal scaling of routers, ensuring that the system can easily accommodate millions of users and their shopping lists.

5 Server Component

This component will be responsible for storing, managing, and replicating shopping list data across a distributed network of servers. We took inspiration from Amazon Dynamo architecture by creating a ring of servers. This ring structure is designed for horizontal scalability, high availability, and efficient data distribution. Each server node that belongs to the ring is responsible for specific data partitions.

Each server node will have a NoSQL database[5] to store and manage our application's shopping lists. This choice of database is due to the fact that our objective is to provide high availability and quick data access. The schema-less nature of NoSQL databases allows for flexible data storage and retrieval. To ensure efficient data distribution this component will employ data partitioning allowing for data sharding. Data partitioning prevents hotspots and ensures that requests are evenly distributed across the server ring. As a result our system will be able to handle large volumes of data while maintaining responsiveness.

To achieve high availability and data resilience, our architecture includes data replication. Each shopping list is replicated across multiple servers ensuring that there are redundant copies available in case of server failures or network disruptions.

Our server ring is easily scalable so it is easy to add new server nodes to the ring as the user base expands and the system becomes overloaded. This elasticity allows our system to remain responsive and reliable over time.

6 Conflict Resolution

With concurrent accesses and updates to the shopping lists, conflicts may arise when multiple users modify the same list simultaneously. To maintain data consistency and integrity, it is crucial to resolve these conflicts in a consistent and disciplined manner.

Our system will use Conflict-Free Replicated Data Types (CRDTs) to address data conflicts. CRDTs are data structures that can be concurrently updated lacking the need for conventional locking mechanisms. They provide robust eventual consistency, which ensures that all replicas converge to the same state, despite the presence of concurrent modifications. We represent the shopping list data structure with CRDTs so that users can add, remove, or check off items simultaneously without creating conflicts.

In addition to CRDTs we will also use Vector Clocks to maintain a causal ordering of operations allowing to track the relationships between different modifications to the shopping list. In addition to identifying causality, allowing you to obtain the most recent and relevant update, vector clocks also help with garbage collection as they allow the system to remove outdated versions of the shopping list, thus optimizing storage.

When a conflict arises, the system will use information from both CRDTs and vector clocks to follow a protocol that allows it to resolve the conflict in a consistent manner. Our conflict resolution mechanisms will work seamlessly in the background, ensuring that users experience a responsive and consistent application, regardless of concurrent updates.

References

[1] Amazon Dynamo paper

[2] Local First: `https://www.inkandswitch.com/local-first/`

[3] CRDTs: `https://crdt.tech/`

[4] Vector clocks: `https://medium.com/big-data-processing/vector-clocks-182007060193`

[5] NoSQL database : `https://www.mongodb.com/nosql-explained`