A local-first Shopping List Application

António Santos, up202008004
Pedro Nunes, up202004714
Pedro Silva, up202004985
José Castro, up202006963

Problem Description & Requirements

Shopping list application** that not only **runs & saves data locally** but contains a **cloud component** to **share lists** with other users and provide **backup storage

Contemplates:

- **Creation & deletion** of **ID-specific** shopping lists
- Users can **add & delete** products if they know a list's ID
- Products in shopping lists have a **quantity**
- Users can concurrently change a shopping list, which outlines the need for **Conflict-Free Replicated Data Types (CRDTs)**
- Cloud-side architecture is expected to handle millions of users, which can lead to **access data bottlenecks**
- **Data** can be **sharded**
- Users are able to use the application **offline** and communicate with a cloud service when online

Client

Built using **JavaScript** and **SvelteKit**, which allowed for faster and overall better development.

Using a **web interface** instead of a CLI is provides a better **user experience**, and more in line with current local-first software practices.

In order to support offline operation of the application we are using **Service Workers.**

This approach makes sharing of shopping list **codes** much easier, since their IDs are just URL parameters, and updates to the application are distributed automatically by accessing the website with a connection.

Local storage of data using **IndexedDB** - more on that later.

Load balancer

We are using **nginx** for load balancing purposes.

A **custom** implementation was possible, but we opted for a more **robust** solution.

This helped us focus on critical points of the application: implementing CRDTs with the shopping lists.

Limitation:

Configuration of newly added servers for the load balancer has to be done manually.

Cloud

Made in **Java** using **Spring Boot**, allowing for easy request handling and server management

Receives:

- **requests for shopping lists** and fetches them from the database
- **incoming changes to database** and takes care of **merging them** (through *CRDT* usage)

Manages:

- **server and database** instances
- **consistent hashing**

Database

Database Sharding: data is stored across a variable number of nodes

Consistent hashing is used to guarantee an **even distribution of data** across nodes

Shopping lists are replicated across 3 nodes in order to improve **availability**

Nodes can be added or removed and shopping lists are redistributed accordingly

Local solution: **IDB-Keyval**

- Very basic implementation on top of **IndexedDB** - Low-level API for **client-side storage**
- Only **one** additional import required - easy **setup**
- The most adequate solution given our chosen **web-based structure** for the client-side of the application

Conflict-Free Replicated Data Types (CRDTs)

Add-Wins Observed-Remove Map: allows for additions and removals.

- Maps [**product_id, dot_value**] -> **Product**
- Retains a **Dot Context**

Causal Counter: to be used in conjunction with the AWORMap.

- Maps [**dot_id, dot_value**] -> **value**
- Retains a **Dot Context**

Dot Context:

- Contains a **causal context**: maps **dot_id** -> **dot_value**
- ...and a **dot cloud**: set of [**dot_id** -> **dot_value**]

NOTES:

- While **delta** usage was disregarded, its implementation is an easy next-step on top of our data types.
- Both a **GCounter** and **PNCounter** were implemented previously, but were not used in the final implementation.

Our Solution - Limitations

Horizontal scaling is a **manual** process - **adding/removing a server** to the application is not straightforward since our load balancer is **not dynamic**, and no **automation** of that process was done.

The number of **virtual nodes** and **replicas** is **hard coded** - these values affect the performance of our application depending on the **number of servers**, as we might want more or less virtual nodes/replicas.

If a server goes **down**, no process has been implemented for **detecting** and **restarting** that server, which makes maintenance a **longer** and more **manual** task.

Local application has no **polling**, meaning changes have to be pushed through an obligatory “Push to Server” button. While implementing polling isn’t hard to do, we felt it was simply unneeded.