U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Smart Distributed List Engine

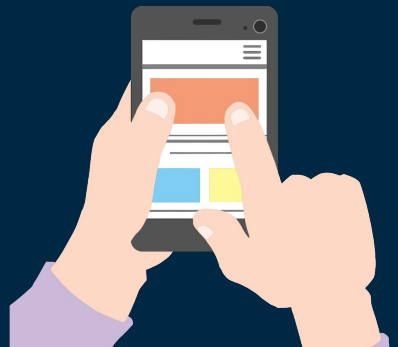
João Pereira up202007145
Mariana Rocha up202004656
Nuno Pereira up202007865

Introduction

SDLE - Smart Distributed List Engine is a local-first user-friendly web app for **collaborative shopping lists**.

It allows users to **create** and **share** lists with family, friends and colleagues, providing simultaneous editing.

Inside a list a user can add items and change its quantity over time as well as deleting it.



Technologies Used

Backend

Kotlin

KTor

Gradle

Frontend

Svelte+SvelteKit

TypeScript

TailWind

Container

Docker

Local Component

Shopping List interactions take place on the **client side**, enabling users to edit information even when **offline**.

CRDTs are used to provide consistency and handle conflicts.

The local component allows users to:

- Create lists;
- Delete lists;
- Change lists name;
- Create items inside of lists;
- Change the quantity of the items inside of lists;
- Change name of items inside of lists;
- Remove items from the list.

CRDTs Implementation

Our CRDT implementation includes:

- **AWSet** → used to implement other CRDTs (AWMap, CCounter, MVRegister);
- **AWMap** → used to implement the list of items;
- **CCounter** → used to register the quantity of the items;
- **DotsContext** → used to implement other CRDTs;
- **GCounter** → implemented, but not used.
- **GSet** →implemented, but not used.
- **MVRegister** → used to store the name of the items and of the lists;
- **PN-Counter** → implemented, but not used.

Synchronization Process

Local-first applications present synchronization challenges since they can operate in both offline and online modes.

The usage of CDRTs is beneficial as they enable automatic conflict resolution, ensuring continuous data synchronization, handling concurrent updates in disconnected or sporadically connected environments and enhancing data reliability.

Besides this, we use Service Workers on the client side to provide automatic synchronization of lists to the backend service.

Server-Side Architecture

Our server-side architecture employs consistent hashing for distributing shopping list responsibilities across all cluster nodes, using virtual nodes to balance data distribution and reduce bottlenecks.

Every node is part of a ring and (using eventual consistency on the cluster view) can reach any peer on the ring, ensuring a distributed storage scheme for shopping lists.

A gossip-based protocol is used to propagate and synchronize the cluster view across all nodes, ensuring that eventually all nodes know about each other. Nodes can be dynamically added to a cluster and the node that processed the join request eventually propagates that information to the rest of the nodes.

Challenges

- Time management;
- CRDTs paradigma;
- Thinking in a distributed way (the Dynamo paper provided a lot of help);
- Problems with network communication between nodes after containerizing them;
- Nothing was easy.

Future Work

- Performance and security concerns.
- Improving the gossip-based algorithm for cluster view propagation (currently some invalid IP addresses are sent in synchronization requests that cause some errors on the nodes that receive them, this does not always happen)
- Make hinted handoff behaviour work not on the node that issues a replication but on the "backup" node receiving the hinted replica.
- Implement data redistribution on cluster topology changes (node addition and removal)
- Better error handling. Better API Interface design.
- Add a way for the user to disable automatic synchronization. Improve this mechanism (as shown in the video, we had to manually reload one of the tabs

Conclusions

This project introduced us to interesting topics such as local-first apps and CRDTs, which were unfamiliar to us.

Additionally, we had the opportunity to explore topics such as gossip protocols, consistent hashing, and other related concepts.

This experience not only expanded our understanding but also captivated our interest on topics related to database management for highly distributed systems.