Shopping Lists on the Cloud

Group 12

Table of contents

01

Requirements

Project Description

This project focuses on developing a **local-first shopping list application**. The system combines **local storage** that allow working offline and a **cloud component** to share data among users and provide backup storage.

Features

- **Create** a new shopping list with a **unique ID**
- **Delete** a shopping list with a **unique ID**
- **Modify** a shopping list with a **unique ID**
  - **Add** item
  - **Delete** an item
  - **Increment** quantity of an item
  - **Decrement** quantity of an item
  - **Acquire** an item
  - **Not acquire** an item

Observations

- Users can concurrently change the list and we aim for **high availability**
- We should carefully design the architecture to **avoid data access bottlenecks**

02

Technical Solution

Shopping List

We have a shopping list entity that **contains** all necessary information and methods to implements previously mentioned **functionalities**.

Attributes

- ID
- creator
- deleted
- items (CRDT)

Methods

- Delete List
- Add/Delete item
- Increment/Decrement quantity of an item
- Not acquire/Acquire an item
- Merge
- ...

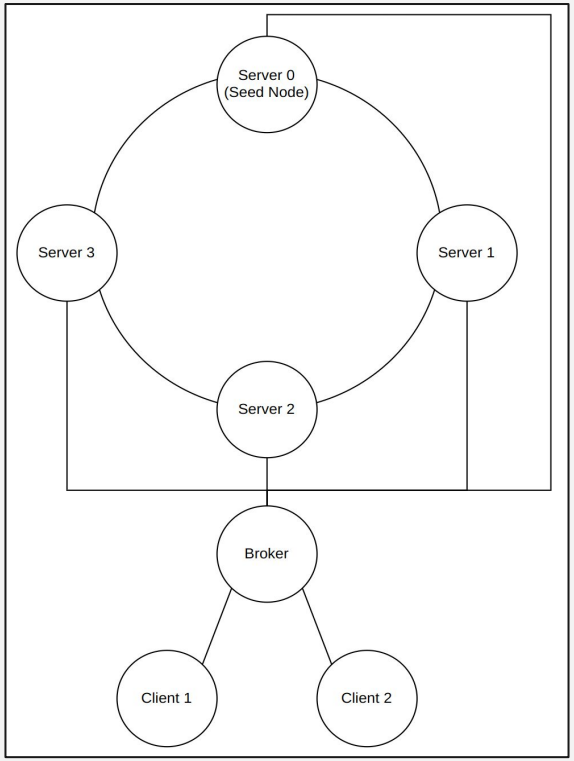
Client side

The user logs in and **enters a list ID**. If the list exists locally, it is displayed; otherwise, a new list is created.

Users can then **perform various actions** such as deleting the list, synchronizing it, or modifying it (adding/deleting items, ...). After each action, the list's state is **saved to a local file** for persistence.

Additionally, the application attempts to **connect to the cloud** and send the list's state after each action. If new information is received from the server, the local list is updated accordingly.

Architecture

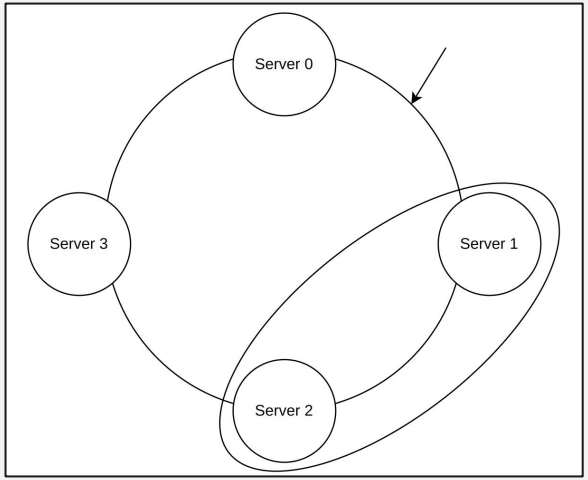


ZeroMQ

- **Client:** REQ
- **Broker:**
  - Client: ROUTER
  - Server: DEALER
- **Server:**
  - Broker: DEALER
  - Another Server:
    - Seed Nodes communication: REQ/REP
    - Forward request to a server in preference list: REQ/REP
    - Propagate updates of a list to servers in preference list: REQ/REP

Distributed data

- **Consistent hashing:** map every node and shopping list MD5 hash in a ring
- **Preference list:** nodes responsible for storing that shopping list
- **Virtual nodes:** same physical node can be placed in multiple positions within the ring to distribute the load more evenly



Server side

Upon **startup**, a server reads a static configuration to obtain the seed node's address. It then sends its information to the **seed node**, which in turn provides details of the remaining nodes.

When a request arrives, the server **hashes** the list ID to calculate a **preference list**. If the server is included in this list, it becomes the **coordinator**.

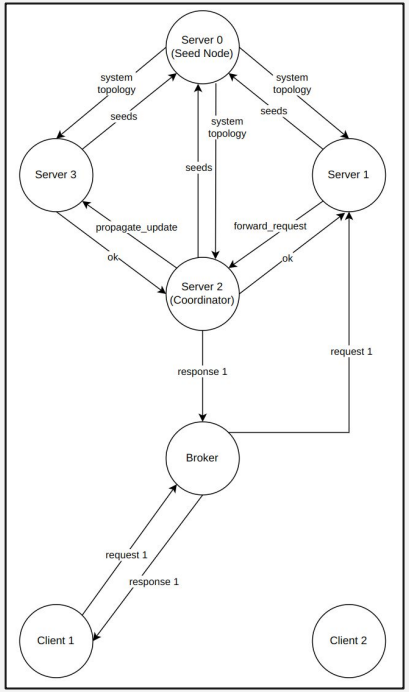
Otherwise, the request is **forwarded** to the first node in the preference list, and subsequently to the next if there's no response, and so on.

Once a server becomes the coordinator, it **merges** the requested list with its stored version.

The updated list is then **propagated** to other **nodes** in the preference list and to the **client**.

Additionally, if a node responds with a **newer version** of the list during propagation, the coordinator **updates** its state accordingly.

To ensure data **persistence**, each server saves its list updates to a local file.



CRDT

- **AWORMap<item_name, Item>**
- **Item:**
  - **CCounter**
  - **EWFlag**
- **AWORSet**
- **DotKernel**
- **DotContext**

03

Demo

04

Questions