U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Shopping Lists on the Cloud

Large Scale Distributed Systems – MEIC03

Ana Rita Oliveira – up202004155

Jorge Sousa – up202006140

Mariana Teixeira –up201905705

Matilde Silva – up202007928

Index

- Requirements and Problem Description
- Implementation Details
- Technical Solution

Requirements and Problem Description

- Local-first shopping list application.
- Data storage, both locally and on the cloud.
- List manipulation – CRUD.
- List sharing between clients.
- Conflict solving with CRDTs.
- Create and delete items.
- Scalable and highly-available system.

Implementation Details

- In this project, we used the following technologies:
  - **React**, for the frontend
  - **Go**, for the client's backend and the server
  - **SQLite**, for the databases
  - **RabbitMQ** (with AMQP), for the client-orchestrator communication
  - **TCP/IP**, for server-orchestrator communication

Client

- Since this is a **Local-First Application**, the client was designed to have data that persists locally.
- This also allows the user to use the app without being connected, i.e., when the user is offline
- In terms of frontend, we opted to have upload and fetch buttons that allow the user to decide when to retrieve and send data from and to the server.

Data Partitioning and Distribution

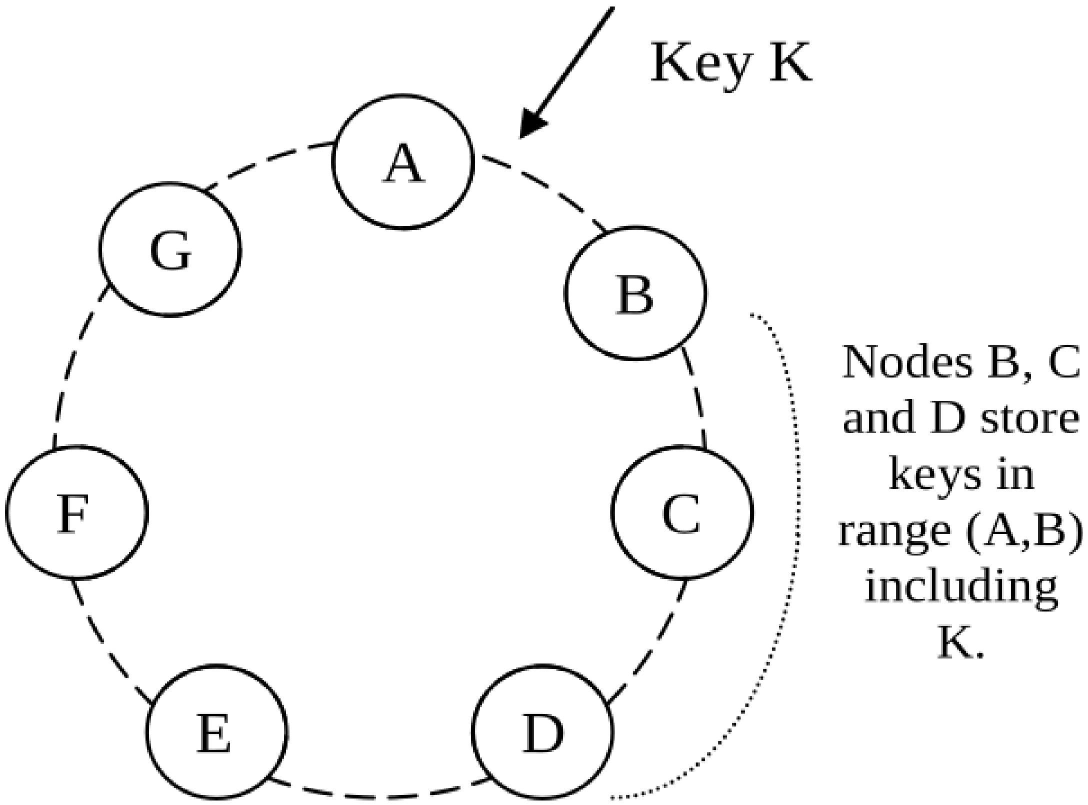

Fig 1: Partitioning and replicatioon of keys in Dynamo ring

- Inspired by the Amazon Dynamo paper, **Consistent Hashing** was used to distribute the Shopping Lists between servers.
- We also used **Virtual Nodes** to ensure even load distribution
- In terms of **Fault Tolerance**, when one or more nodes fail, messages are redirected to the next available node in the hash ring

Server

- A **Multi-Server** solution with **Quorum Consensus** was used to ensure **high availability**
- This solution allows for a **scalable system**
- It is always listening for possible **quorum participation requests** initiated by other servers.
- When the server initiates a quorum the **minimum number** of participants is half the number of active servers plus one.

Orchestrator

- The orchestrator acts as a **middle-man** between the clients and the servers.
- It is in charge of the message **re-routing** and even distribution of requests between each server. In other words, it mandates de Hash Ring. Note that, the used hash is MD5.

CRDT – LexCounter

- The **LexCounter** CRDT, similar to PNCounter, allows for both increments and decrements in the quantity of an item.
- This CRDT contains:
  - Id
  - Lexicographic Pairs
- Present both in the Client and the Server