# wishCloud: Local-First Shopping List Application

Gonçalo Ferreira
up202004761

Ricardo Ribeiro
up202310095

Vítor Cavaleiro
up202004724

November 24, 2023

## 1 Introduction

The aim of this project is to address the challenges and requirements associated with the design and implementation of a local first shopping list application.

The challenges associated with this type of system are many. Key issues include offline usage, user experience, data storage, backup and recovery, communication middleware, data synchronisation, conflict resolution, scalability and performance. Many of these challenges are analogous to those outlined in the Amazon Dynamo paper [2], so we have adopted some of their solutions in our proposed system.

## 2 Architecture

The architecture of the local-first shopping list application is designed to meet the primary objectives of the project. It combines local and cloud components to ensure data availability, consistency and a great user experience.

The core principle of our architecture is the local-first approach. Users can manage their shopping lists on their local devices, providing a responsive and offline-capable experience. These local devices are capable of connecting directly to cloud-based servers, letting users access and modify their shopping lists in cooperation with others.

In the cloud layer, inspired by Amazon Dynamo, we ensure high availability, data synchronisation and conflict resolution for online users. Data consistency, scalability and fault tolerance are supported by this cloud-based infrastructure.

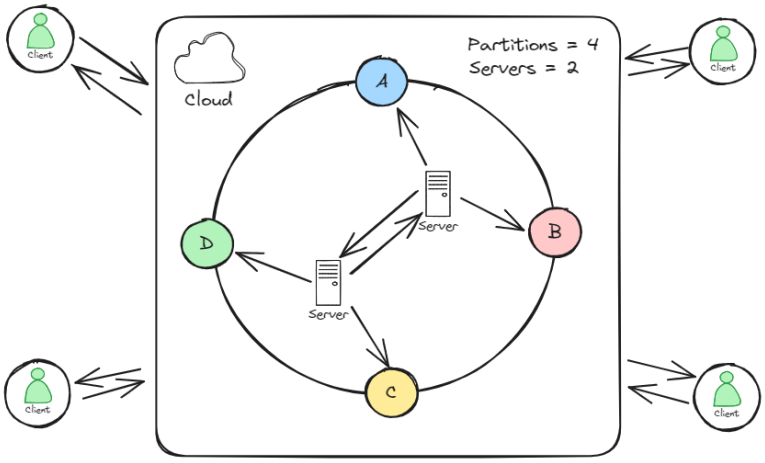


Figure 1: Architecture Overview

## 3 Cloud Design

The cloud is the infrastructure that provides a backup for user data and the ability to share shopping lists. The system aims to be used under heavy conditions, avoid bottlenecks, and achieve data persistence even after server failure. Data should be replicated among the servers and distributed in a way that avoids hotspots. Dynamically adding nodes to the cloud is also a key part of the design to fulfil the growing demand for the system.

Consistent hashing provides dynamic addition of new nodes without affecting the whole system by facilitating partition assignment. The implementation of virtual nodes/tokens to map servers in the ring system also provides good load distribution. The fine-tuning of the previous aspect will depend on how partition sizes are defined and how many tokens are assigned per server. The most efficient way would be to assign equal-sized Q partitions in the hash space and provide Q/S tokens per server, S being the number of servers currently active in the cloud. To assure data persistency, whenever an item is assigned a coordinator node, it is responsible for replicating the shopping list on the following N - 1 tokens, N being the number of tokens the item needs to be written.

The system implements a consistency protocol, with a configurable W(writes) and R(reads). The change in these values can affect both consistency, availability and persistence.

In case of node failure, and to enforce the necessary writes and reads, hinted handoff will be applied so that no replicas are lost. In case a server is unavailable, the following healthy nodes on the ring will store the replica. Once the original server is back online, the "backup" server will deliver the item and delete it from its local database, always assuring a consistent number of replicas in the ring. If the backup server is lost before returning to the original one, then there needs to be an anti-entropy protocol to keep replicas synchronised. Merkle trees offer an effective solution.

When initialising the system, there needs to be seeds, in this case, statically configured servers, that all newly created nodes know before joining the ring. Every new instance maps its corresponding tokens on the hash space. Finally, it contacts the seeds to let the ring know a new server joined.

The information about the newly added nodes or their departure, as well as node failures, is propagated in the system through a gossip-based protocol. Every second, nodes contact at random another pear in the system, to keep information updated regarding changes in the ring and failures that occurred.

Regarding local data consistency, each server will have a SQLite or BDB database.

It will also need to implement an API to contact other servers and receive commands from the user.

# 4 Client Application

Our client-side design is based on the local-first principle, which ensures that data is stored primarily on the client's device. This approach allows users to access and modify their shopping lists even when offline, eliminating the need to connect to the cloud for proper operation. To achieve this, we use a compact and fast local database (SQLite) that allows users to interact with their shopping lists and then synchronise changes with the cloud-based servers.

In line with the local-first approach, our priority is to develop a user-friendly interface. This interface allows users to create and share shopping lists using a unique ID. This ID is created by the client using the Universally Unique IDentifier (UUID), which is compliant with RFC 4122 [1]. This makes collaboration much easier, as multiple users with the same ID can modify shopping lists simultaneously. Key features include the ability to add products, update product information, mark items as purchased and specify the quantity of each product.

The client-side application is designed to connect directly to the cloud-based server. No intermediate routing is required. When the client is first initiated, it will have hard-coded the seeds of the cloud ring to connect to for the first time. Every 10 seconds, clients poll a random node for updated ring information. Changes are synchronised with the cloud-based server when the Internet connection is re-established, ensuring data consistency across all client devices.

Our design takes advantage of the natural load-balancing properties of partitioning across the server ring, providing an efficient and balanced data handling system. Clients are aware of the server ring partitions, enabling faster and more efficient operations in the cloud. As users interact with shopping lists, client-side routers determine which server to target for their requests, streamlining data handling.

# 5 Conflict Resolution

Conflict resolution is a critical aspect of the application, ensuring that data remains consistent and accurate, even when multiple users are editing the same shopping list. We will explore conflict resolution strategies, including the transition from Last-Writer-Wins to Conflict-free Replicated Data Types (CRDTs) [3].

CRDTs are innovative data structures that allow concurrent updates without the classic locking mechanisms, making them ideal for collaborative environments. These structures ensure eventual consistency, where all replicas of the shopping list data align over time, even with simultaneous modifications. This feature is crucial for a smooth user experience in shared shopping lists, as it permits users to simultaneously add, remove, or check off items without causing data conflicts.

In order to assure causal order for the CRDTs, we will also implement vector clocks. Vector Clocks provide a chronological context to each operation, helping us trace the sequence and interrelation of various changes made

to the list. This combination offers a comprehensive solution to conflict resolution in real-time collaborative applications.

In the event of a conflict, our system adeptly utilises the synergy of CRDTs and Vector Clocks to apply a consistent and efficient conflict resolution protocol. This advancement not only improves data integrity but also enhances user collaboration, making our application more reliable and user-friendly for managing shared shopping lists.

# 6 Conclusion

The architectural design of the "wishCloud" local-first shopping list application is crafted to provide a robust, scalable, and consistent solution for collaborative shopping list management. It empowers users to manage their shopping lists effortlessly while maintaining data integrity, privacy, and availability. As the number of users grows, the system can adapt seamlessly, ensuring a consistently positive user experience.

# References


[1] *A Universally Unique IDentifier (UUID) URN Namespace*. 2005. URL: https://www.rfc-editor.org/rfc/rfc4122.

[2] *Dynamo: Amazon's Highly Available Key-value Store*. URL: https://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf.

[3] Nuno Preguiça, Carlos Baquero, and Marc Shapiro. "Conflict-Free Replicated Data Types (CRDTs)". In: *Encyclopedia of Big Data Technologies*. Springer, May 2018. DOI: 10.1007/978-3-319-63962-8_185-1. eprint: 1805.06358.