Architecture Of A Local-First Highly Durable Application

Abilio Epalanga
up202300492@edu.fe.up.pt

Daniel Rodrigues
up202006562@edu.fe.up.pt

Martim Videira
up202006289@edu.fe.up.pt

Miguel Silva
up202007972@edu.fe.up.pt

Faculdade De Engenharia da Universidade do Porto

Abstract

This paper documents and discusses the architecture choices made to create a Highly Durable Collaborative Local-First Shopping Application.

1 Local First Application

1.1 Ownership

Having full ownership of our work was something that we as a society were so accustomed to in the age of pen and paper that we failed to realize that companies have been holding our data hostage since the beginning of the Internet Revolution.

In local first applications, users have full ownership of their data, as it should be: in a standard Open Source format; stored in the user device; easily manipulated

1.2 Offline Support

While commuting on the Metro of Porto, one is faced with low Internet speeds and even unstable connections. Most applications stop working or hinder the user's workflow as soon as it comes to these situations. Even Overleaf, a Browser Based Editor stopped us from writing this document in reaction to FEUP's Library Internet.

In Local First Apps the system is designed to be offline as a rule and not an exception. So offline support will be granted as a default and not as a nice-to-have.

1.3 Progressive Web Application

If the user should be able to freely tinker with their data, the software that manages and creates the information should be free, open-source, platform-independent, and easily distributed.

With these constraints in mind, the team researched several philosophies like building native platforms and creating normal web applications from scratch, to understand what would fit our project the best. These possibilities came with drawbacks such as having multiple code bases and being hard to distribute - as you would need to resort to installers, app stores, and sharing code through git or docker.

The simplest solution that meets the requirements is a Progressive Web App that can run on any device that has access to a modern Internet browser - which fulfills our cross-platform requirement. It is also installable, for offline use, meeting the Offline Support requirement for Local First Applications and it should be accessed and then downloaded by just accessing a URL - making its distribution rather easy, as required.

2 Collaboration

Each shopping list should be editable by multiple users, with the guarantee that their changes will eventually converge to the same state (Figure 5).

2.1 Conflict-Free Replicated Data Types

With CRDTs, applications can have users making changes concurrently on their local devices. If they have a way of communicating - which will be referred to as having an internet connection - then they can share their respective shopping lists as CRDTs and merge them to update both states.

Using this technology it is possible to process concurrent changes and allow the system to have the highest possible availability (working offline) while having the power of collaboration when a connection is established.

This decision comes with the trade-off of strong consistency - something that developers have been accustomed to with ACID SQL-like databases- for eventual consistency.

3 Data Model

Each list will be stored in a file on the user's local disk with the file name being a Unique ID. The contents of that file will be a CRDT representing the list.

3.1 Workflow

When a new list is created it is assigned a new UID. A CRDT representing an empty shopping list is also created and stored to disk. The user can now edit their shopping list freely, without caring about technical aspects like the speed of their internet connection or if they're even connected to the storage server - as synching with the storage server happens in the background.

Users can then share links - a standard URL that contains a shopping list UID to ease the collaboration between users.

If a user without connection accesses the URL of a shopping list given by another user, he won't face any problems or warnings alerting that the provided list can't be reached. Instead, a new empty list with that same UID will be created so the user can proceed to edit as usual. The contents of the list will be updated and fetched whenever the connection is established again (Figure 5).

3.2 API

Our server will have 2 endpoints.

- An endpoint that serves our Progressive Web App (The Client) (GET REQUEST) (Figure 5)

- A synching endpoint that takes as a parameter a shopping list to synchronize (UID, CRDT) with the server. This merges the user's changes and returns the same UID and the updated CRDT so the user can merge locally and get the updates that other users might have done to that list (POST REQUEST).

4 The Cloud

What is "Cloud" doing in a Local-First paper?

In local first development instead of looking at the cloud as a source of truth, we can see it as just another peer that anyone has easy access to (not needing to circumvent NAT).

So we use it as a synching and sharing site.

4.1 The Dynamo Paper

In the following subsections, it'll be discussed techniques implemented on Dynamo from which the team took inspiration. From partitioning to failure detection, the way the server side is structured is in various ways similar to the Dynamo architecture.

4.2 Partitioning

To deal with a large amount of requests, there comes the need for load balancing and subsequent partitioning of the server-side instances. Parallelism is a must in the discussed architecture. The team suggests implementing **consistent hashing** of each UID and having different partitions holding different ranges of hashes, with the help of a **preference list** to understand to which partitions should the intermediate server redirect in case of failure.

This intermediate server should be the one between the clients and the remaining partitions and is responsible for redirecting and managing the load of requests (Figure 5).

4.3 Replication

Within each server-side partition, it should contain not only its range of hashes but its neighbors' ranges as well. The concept of *neighbor* comes from the consistent hashing implementation. The divisions that make up the different ranges indicate that for each range, there are two immediate neighbors.

In this project's architecture, for every partition, the team plans on storing replicas in their immediate neighbors, **adding** them to the server's **preference list**.

4.4 Writing Quorum

With the Local-First App philosophy being used, the team believes that a write quorum of 1 will grant the highest availability and also great durability as there will always be more replicas alive in the user's device. So even if the node that was written to malfunctioned, as the client is the owner of the data, no real data loss would happen.

With a Quorum of 1, a sync operation will not be successful only if all nodes are unavailable.

4.5 Failure Detection

Like Dynamo, a gossip-based protocol will be used that has a negative assumption that if a node doesn't respond to another node's message in a set amount of time, this node is assumed to have failed.

Therefore all the requests that the starting node would make to the node it assumed to fail will be re-routed to other nodes down the preference list.

Nodes that have data belonging to other nodes will periodically try to move it to the correct nodes. So eventually the system will be balanced.

4.6 Adding/ Removing Nodes

These changes could only be possible via the administrator's CLI. If a given partition is removed from the server composition, its immediate neighbors become responsible for the removed partition's range of hashes. In case the node is added back to the production environment, it takes a pre-determined position based on the hashes that already exist.

5 Discussion and Further Work

Overall, the team is satisfied and excited to implement the resulting architecture. The way the endpoints are organized and the simple way the team makes use of CRDTs (always with the use of POST request) to merge different versions was a by-product of many discussions there were had over the last two weeks. The team's main goal was to define a working solution that could be used by as many people as possible, which resulted in the Progressive Web App idea. By reading the Dynamo paper, all of the members gained insight on how the server should be structured, which also came in great help.

Other than that, it is now crucial to decide on which tech stack the team will be using to develop this project, to make a fully functional, scalable, and available server. By making (possibly) making use of parallelism and concurrency to manage the load of server requests, on top of an event-driven, asynchronous methodology.

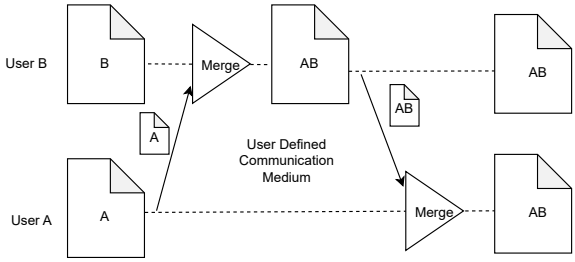

Figure 1: Both Clients Sync

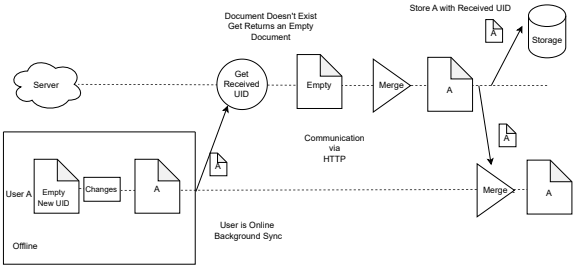

Figure 2: Creating A New Shopping List

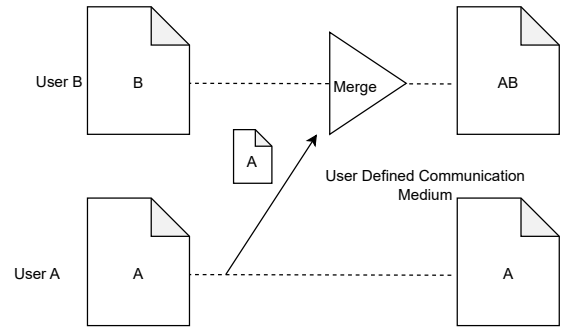

Figure 3: A Client Syncs

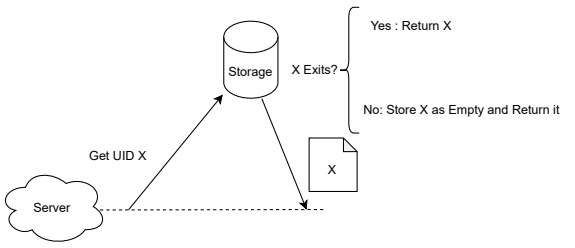

Figure 4: Server Get UID