Local First Shopping List using CRDTs

Adilson Silva
up202212352@up.pt
Faculty of Engineering of University of Porto
Porto, Portugal

Sofia Teixeira
up201806629@up.pt
Faculty of Engineering of University of Porto
Porto, Portugal

Luís Osório
up202004653@up.pt
Faculty of Engineering of University of Porto
Porto, Portugal

Luís Morais
up200800621@up.pt
Faculty of Engineering of University of Porto
Porto, Portugal

ABSTRACT

This article elaborates upon the intended design of a local first shopping list platform, using Conflict-Free Replicated Data Types (CRDT), developed for the Large Scale Distributed Systems (SDLE) class.

The platform should allow for a user to modify a shopping list, identified with a unique hash, by supporting the removal and addition of products, as well as the creation of new listings.

1 INTRODUCTION

The goal of this project is to develop a shopping list platform, calling attention to local storage as its main way of saving changes, saving to the cloud at select moments so as to allow for conveniences such as offline use, assured data ownership and less latency due to the reduced number of round trips made to the servers. The platform is built by taking advantage of Conflict-Free Replicated Data Types as a way to allow for concurrent updates to the shopping lists while still guaranteeing consistency.

A load balancer is integrated into its architecture along with a cloud design inspired by the well known and highly regarded Amazon Dynamo's architecture.

2 APPLICATION

As a shopping list platform, a client should have the possibility of creating lists, which exist on the cloud and are fetched with their unique identifier through the use of, for example, an URL, until a client decides on their deletion. Inside of a shopping list, an user can add new items to the list as they see fit, specifying the initial amount, which will then be added to the list. From that point on, like other items already present, the client can increment or decrement their amount as they please.

This list will eventually be synced to the cloud when an internet connection is established where conflicts are internally handled. The client will also have the option of pulling the latest version of the list from the cloud and seeing the changes that it underwent.

3 CRDTS

3.1 Shopping List - CRDMap

A list is implemented as a Conflict-Free Replicated Data Map (**CRDMap**) mapping keys, implemented as strings, to Conflict-Free Replicated Items (**CRDItem**).

Our **CRDMap** exposes four operations, at the least:

- *put(key, item)*, which maps an item to a key
- *get(key)*, fetching items mapped to the given key
- *remove(key)*, removing the items mapped to the given key
- *update(key, operation)*, which runs the given operation on the item mapped to the given key

The *update()* operation is special in the sense that the items to which the keys are mapped to are CRDTs themselves and this update could mean the update of the embedded CRDT's state.

Changes utilize causal context[1] as a method of data versioning and handling of divergent data from distinct users

This design for the aforementioned CRDT is largely on the definition of a CRDT Map found here, in the section ***2.1.5*** and implementation uses the ORMap as an inspiration.

3.2 Shopping Item - CRDItem

Items contained in the shopping lists can be one of two possible CRDTs:

- A **CCounter**, which is a counter supporting *increment()*, an operation that by default increments by 1 if no value is given, and *decrement()*, which behaves contrarily to the increment operation by decrementing by 1 if no value is given. Changes are handled in way that when it is incremented/decremented, "it consults its last updated count, changes it and stores it under a new dot entry after deleting its previous entry"[2]
- An **EWFlag**, an **Enable-Wins Flag** that handles concurrent changes by prioritizing the change enabling the flag.

While items could be implemented simply using counters, for items only meant to be bought once, the **CCounter** CRDT is a little too bulky and has a larger overhead when being transmitted to the servers, whilst the **EWFlags** are more lightweight.

4 CLOUD ARCHITECTURE

Cloud-side architecture, much in the same way that the Amazon Dynamo is designed as decentralized, lessening, or even possibly avoiding, damage in the case of server failures while at the same time allowing for easy scaling with the addition of new servers.

Users will, when available, connect to a load balancer responsible for determining the node most appropriate for each user so as to ensure the best possible performance from the platform.

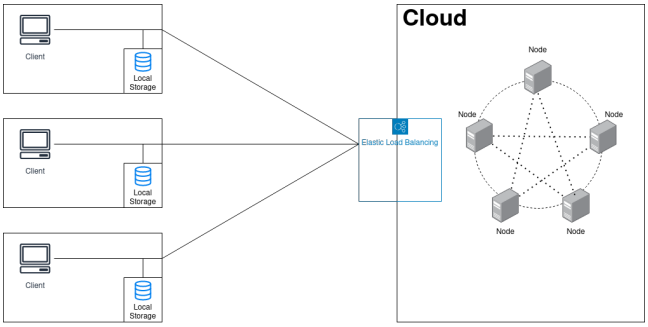

Figure 1: Cloud architecture

4.1 Data Storage

Inspired heavily by the Amazon Dynamo's architecture, shopping lists are stored as key-value pairs, where each key is correspondent to the URL used to access the list and whose mapped value is the instance of the shopping list being retrieved. A hash is applied on the list's key, returning an identifier that will determine what storage node shall be responsible for handling said list.

4.2 Partitioning

Partitioning is performed through the use of the partition mechanism mentioned in the Dynamo's article known as "consistent hashing"[3] where each hash is mapped to a finite and continuous hash space where the highest possible value represents the lowest value of another. Each shopping list is mapped to the server whose first token is higher than the list's hash.

The Dynamo article lists three different strategies to handle this task, of which we chose the **first strategy**, where each server/node handles a hash space proportional to its own processing capabilities.

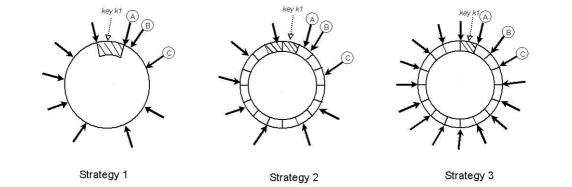

Figure 7: Partitioning and placement of keys in the three strategies. A, B, and C depict the three unique nodes that form the preference list for the key k1 on the consistent hashing ring (N=3). The shaded area indicates the key range for which nodes A, B, and C form the preference list. Dark arrows indicate the token locations for various nodes.

Figure 2: Partition strategies

4.3 Replication

To ensure data availability and durability in the event of system failures, the shopping lists are replicated amongst multiple servers, where each list is replicated at N hosts, where N is parameterized.

The replication of each key is handled by its coordinator node, that is, the node responsible for handling the keys that fall within its range. Each coordinator node will replicate their keys at nodes in a clockwise manner until reaching the N-1th node. This makes it so that each node takes on responsibility for the keys starting from their range all the way to the keys at the end of its Nth successor.

4.4 Data Versioning

While all changes to the shopping lists themselves are handled by their respective CRDT implementations, eventual consistency is employed, again following the Amazon Dynamo's architecture, by asynchronously propagating updates across all nodes. From asynchronous updates arise a lot of possible issues, such as the delay of replication in case of system failures, which leads to inconsistency between replicas. To tackle these issues, a compromise is made between the possibility of losing some updates and ease of implementation by utilizing last-writer-wins policy.

4.5 Server Addition/Removal

Due to the decentralized nature of the platform, replication is necessary, meaning that the addition of new servers/nodes will also imply a need for replicas from other nodes. However, due to the asynchronous nature of the replication strategy, other nodes will automatically connect to new nodes and transmit the necessary data.

4.6 Fault Tolerance

For all the aforementioned sections, fault tolerance is handled through the techniques detailed in the Dynamo's article, be it because of its heavy influence on this project as a whole or due to its known efficiency and correctness.

REFERENCES

[1] Carlos Baquero. 2023. CRDTs: State-based approaches to high availability. https://moodle2324.up.pt/pluginfile.php/162922/mod_resource/content/1/CRDTsDARE2023SDLE.pdf

[2] Ali Shoker Carlos Baquero, Paulo Almeida. 2018. Implementation of CRDTs with delta-mutators. https://github.com/CBaquero/delta-enabled-crdts

[3] Madan Jampani Gunavardhan Kakulapati Avinash Lakshman Alex Pilchin Swaminathan Sivasubramanian Peter Vosshall Giuseppe DeCandia, Deniz Hastorun and Werner Vogels. 2007. Dynamo: Amazon's Highly Available Key-value Store. https://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf