Shopping List on the Cloud - Architecture

Group Name : SyncShopSquad

Francisco Serra - up202007723
João Reis - up202007227
João Teixeira - up202005437
Rui Pires - up202008252

HIGH AVAILABILITY STRATEGY (CRDTs)

Our shopping list project aims to facilitate real-time collaboration among users in a distributed environment. To achieve this, we'll be using a key-value ORMap CRDT structure where the key represents an article, and the value is a CCounter (Causal Counter) representing the quantity of that article. This design choice ensures efficient tracking of item quantities while handling concurrent updates and conflicts in the best way possible.
This way multiple users can concurrently add and remove items from the shared list having the safeguard of the ORMap CRDT that ensures updates are efficiently synchronized, and conflicts are automatically resolved. Users can also change their local copies of the shopping list even if not connected to the internet since when they reconnect the CRDT will reconcile all the offline changes made by all users smoothly.

CLOUD SIDE ARCHITECTURE

Partitioning - Consistent Hashing

In the development of our architecture, we had to consider the possibility of a server being shut down or failing, raising the question of what to do with the data that was stored in that server. To solve this problem, we decided to use consistent hashing. This method is used to solve the problem of data partitioning in a way that ensures even distribution between the servers and maintains availability of the data even in case of failures.
To better understand this concept, we can look at Figure 1.

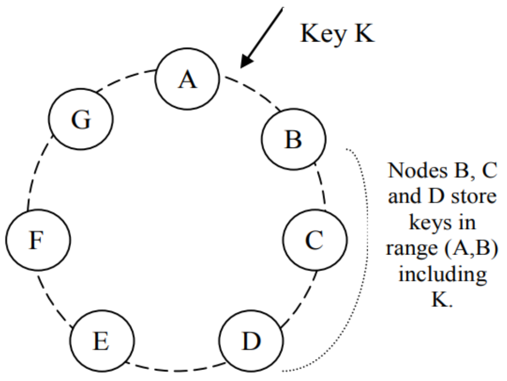

Figure 1: Consistent Hashing

As we can see, the goal is to map data items and nodes to a common identifier space, typically a ring or a circle. Each data item is assigned to a point on the ring, and each node is assigned to multiple points on the ring. The mapping is done using a hash function, which consistently places the data item and node on the ring based on their identifiers. This ensures that when a data item needs to be located, it can be efficiently routed to the appropriate node.

On top of this, we will also be using virtual nodes to improve the distribution of data even further. This concept consists of replacing the single point that each physical node is mapped to in the ring with virtual nodes that represent each physical node multiple times. These virtual nodes act as intermediaries, creating additional entry points on the ring for data distribution, while being mapped to a single identifier on the ring.

Data Replication

To improve the system's fault tolerance we will also be implementing key replication, a method that consists of creating and maintaining duplicate replicas of a specific data item across multiple nodes within a network. These copies will have to be kept in sync and will serve as backups in case one of the nodes is removed from the network. When a node is assigned with a key it should send replicas to an N number of its clockwise successor nodes in the ring, (See Fig.1 where N = 3).

Replica Synchronization Strategy

In our system, data synchronization is a pivotal process implemented for ensuring eventual consistency among nodes. This mechanism involves the periodic exchange of messages from each server to its N replica servers. By systematically transmitting these messages, we facilitate the harmonization of data across nodes, promoting a state of eventual consistency within the system. This approach plays a crucial role in maintaining uniformity and accuracy across replicas, safeguarding against potential discrepancies that may arise over time.

Propagation to Clients

In our system architecture, a load balancer server functions as a central orchestrator, functioning akin to a leader server equipped with comprehensive knowledge of the node ring configuration. This strategic positioning allows the load balancer to efficiently distribute client requests among multiple servers. Each client possesses a personalized awareness of a specific list of IDs relevant to their interests. Leveraging this tailored knowledge, the load balancer dynamically routes requests based on the unique preferences of individual clients. Notably, any modifications to the lists associated with unique IDs trigger real-time notifications from the load balancer to the affected clients, ensuring prompt updates.

Solving Node Failure

For handling node failures, we will implement a straightforward strategy to manage this issue effectively. When a node becomes unresponsive (*i.e* is no longer able to properly communicate or respond to requests from other nodes or components in the network), we will take immediate action to remove it from the active node ring. That means that all the keys belonging to the node that just got removed will need to be reassigned to the next node in the ring.

Subsequently, for each key within the range of the departing node, the replicas are reassigned to the selected successor node and its N-1 clockwise successors in the ring. It is crucial to ensure that the replication factor (N) is maintained during this process.