SHOPPING LISTS ON THE CLOUD

Large Scale Distributed Systems

Ana Beatriz Fontão - up202003574

José Luís Rodrigues - up202008462

Maria Sofia Gonçalves - up202006927

INTRODUCTION

- Creation of a local-first shopping list application.
- Data stored locally and on the Cloud.
- Lists can be shared between clients.
- CRUD operations over the lists' products.
- Conflits are resolved using CRDTs.
- Scalable and highly-available system.

INTRODUCTION

- What is being done?

An application that allows users to create shopping lists on their own devices, while at the same time collaborating on the cloud.

- How is it being done?

By using data versioning, sloppy quorum, consistent hashing, replication, a local-first and cloud-use approach and Conflict-free Replicated Data Types.

- Why is it being done?

To get the best of both worlds: seamless collaboration between users in different parts of the world (cloud) while maintaining full ownership of one's data (local-first).

RELATED WORK

Dynamo: Amazon’s Highly Available Key-value Store.

- data versioning
- sloppy quorum
- consistent hashing
- replication

Conflict-free Replicated Data Types

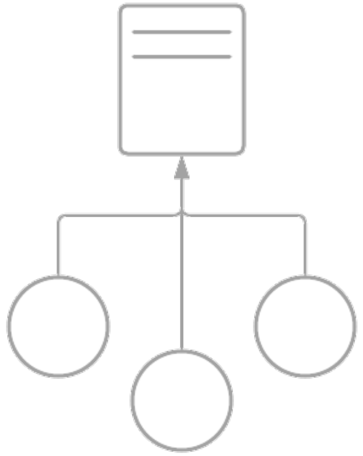
- data type whose replicas always converge, despite failures

Local-first software

- offline functionalities as a primary concern
- incorporates cloud synchronization as a background task

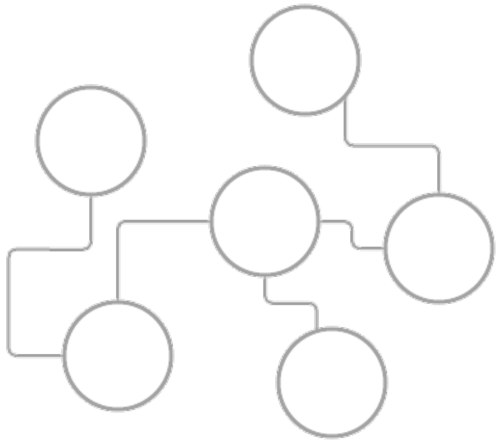
ARCHITECTURE

Single-Server



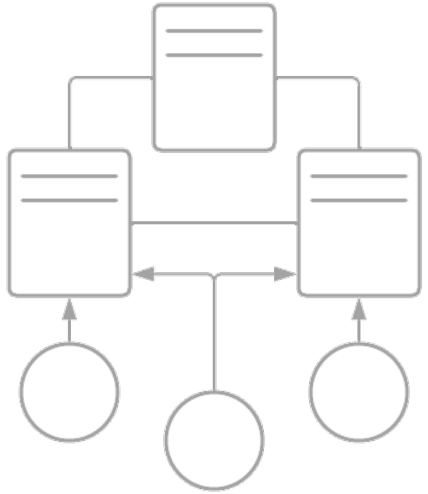
- + Easy to implement
- - Limited to vertical scaling
- - SPF: low availability
- - Vulnerable at increased loads

Peer-to-Peer



- + Highly scalable
- + Limited server interaction
- + Replication is done in the clients
- - Users are prone to disconnect (information gets lost)

Multi-server (our solution)



- + Vertically and Horizontally Scalable
- + High availability (clients can talk to all servers)
- - Loss in consistency (**SEC**)
- - Introduces complexity to ensure synchronization

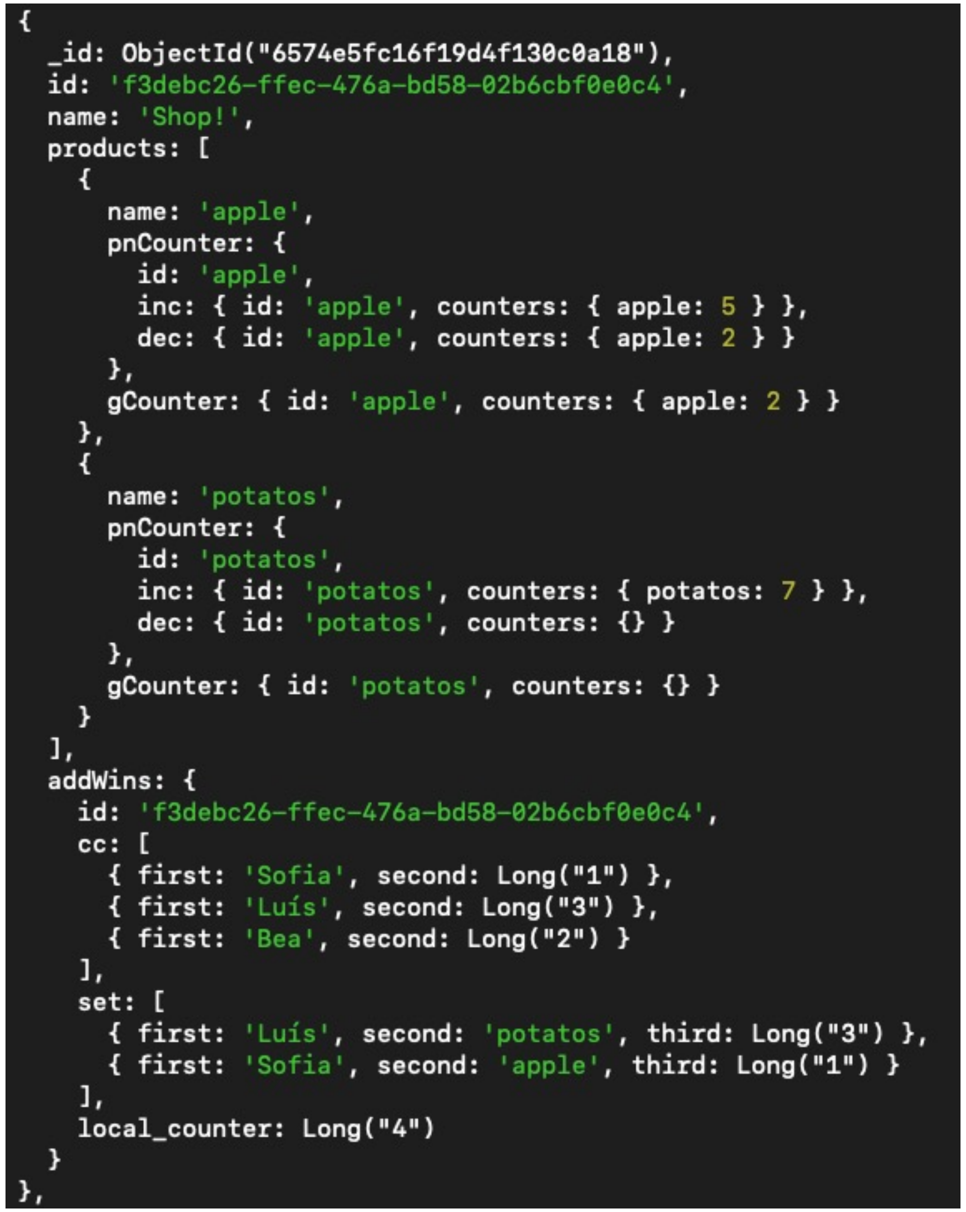
REPLICATION

- Usage of **Quorum Consensus** for reading and writing operations:
  - Lists are replicated at **N** nodes (servers).
  - Write operations are valid after **W** sucessful writes.
  - Read operations are valid after **R** sucessful writes.

- CAP Theorem:
  - Using Quorum increases the availability (A) and partition tolerance (P) of the system.
  - This comes at the cost of consistency (C).
  - Solution: Usage of CRDTs to guarentee **eventual consistency**.

SHOPPING LIST

- The shopping list is composed by the elements:
  - **Id**: a 128-bit identifier for each list, ensuring that each Id is unique;
  - **Name**: the list's name, given by the user that created it;
  - **Products**: a listing of the desired products, explained in the following slides;
  - **Addwins**: composed by its elements, allows the merging of the list's products;
  - **Deleted:** a flag that is set to true if an user, who had access to the lists, decides to delete it.



```
{
  _id: ObjectId("6574e5fc16f19d4f130c0a18"),
  id: 'f3debc26-ffec-476a-bd58-02b6cbf0e0c4',
  name: 'Shop!',
  products: [
    {
      name: 'apple',
      pnCounter: {
        id: 'apple',
        inc: { id: 'apple', counters: { apple: 5 } },
        dec: { id: 'apple', counters: { apple: 2 } }
      },
      gCounter: { id: 'apple', counters: { apple: 2 } }
    },
    {
      name: 'potatos',
      pnCounter: {
        id: 'potatos',
        inc: { id: 'potatos', counters: { potatos: 7 } },
        dec: { id: 'potatos', counters: {} }
      },
      gCounter: { id: 'potatos', counters: {} }
    }
  ],
  addWins: {
    id: 'f3debc26-ffec-476a-bd58-02b6cbf0e0c4',
    cc: [
      { first: 'Sofia', second: Long("1") },
      { first: 'Luís', second: Long("3") },
      { first: 'Bea', second: Long("2") }
    ],
    set: [
      { first: 'Luís', second: 'potatos', third: Long("3") },
      { first: 'Sofia', second: 'apple', third: Long("1") }
    ],
    local_counter: Long("4")
  }
},
```

CONSISTENCY (CRDTS)

- Context-free Replicated data Types (CRDTS) are a data type whose replicas always converge, despite any number of failures that may happen, thus ensuring strong eventual consistency.

- The following structures were used for the project:
  - **Add-wins Set**
  - **PNCounter**
  - **GCounter**

ADD-WINS SET

- The Add-Wins Set is a CRDT that allows concurrent **addition of elements** ensuring consistency and avoiding conflicts.
- It contains three elements:
  - **Local counter**: A client private variable that increases on each operation
  - **Casual context**: Set of tuples that contain the client's id and their respective local counter. This set allows the system to keep track of the change history
  - **Set**: Set of triples that contain the client's id, a name of an element and their respective local counter. This set keeps track of the presence of elements
- **Merges** are performed based on the differences between the casual context and the set.

G-COUNTER (GROW-ONLY)

- The G-Counter is a data structure that tracks the **incremental quantity** of an element.
- It contains two elements:
    - **Id**: Represents the id of the element.
    - **Counter**: Replica variable that can only increase and never decrease.
- The merges involves taking the maximum counter value out of all the replicas, ensuring that the total count is precise and never decreases over time.

PN-COUNTER (POSITIVE-NEGATIVE)

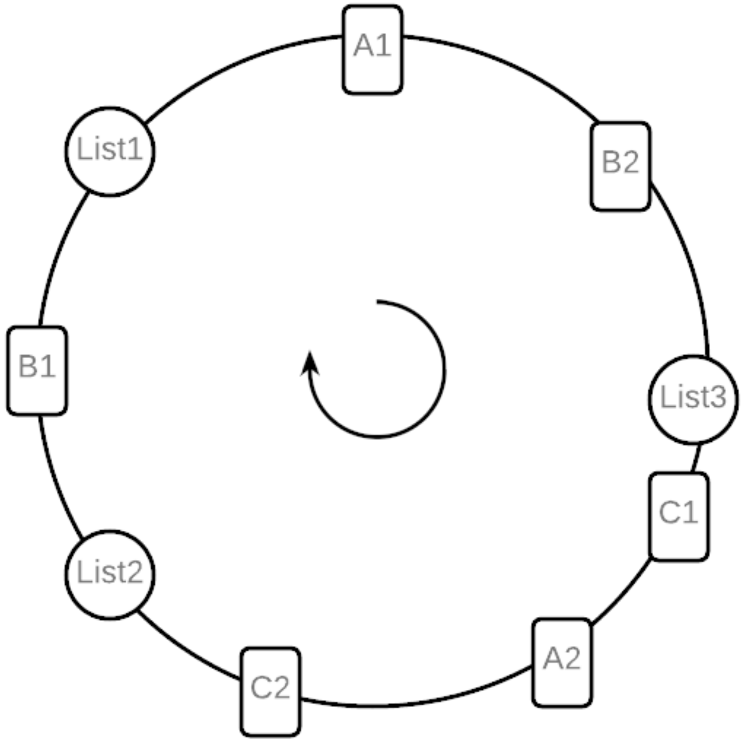
- The PN-Counter is a CRDT similar to the G-Counter. The main difference is that this data structure allows quantities to be **increased and decreased**.
- Contains three elements:
  - **Id**: the product’s name;
  - **Incremental counter**: a GCounter, responsible for informing about the quantity of the product that is added;
  - **Decremental counter**: a GCounter, responsible for informing about the quantity of the product that is removed;
- The merges are performed on each counter.
- The final quantity is given by the **difference** between the **incremental** counter and the **decremental** counter.

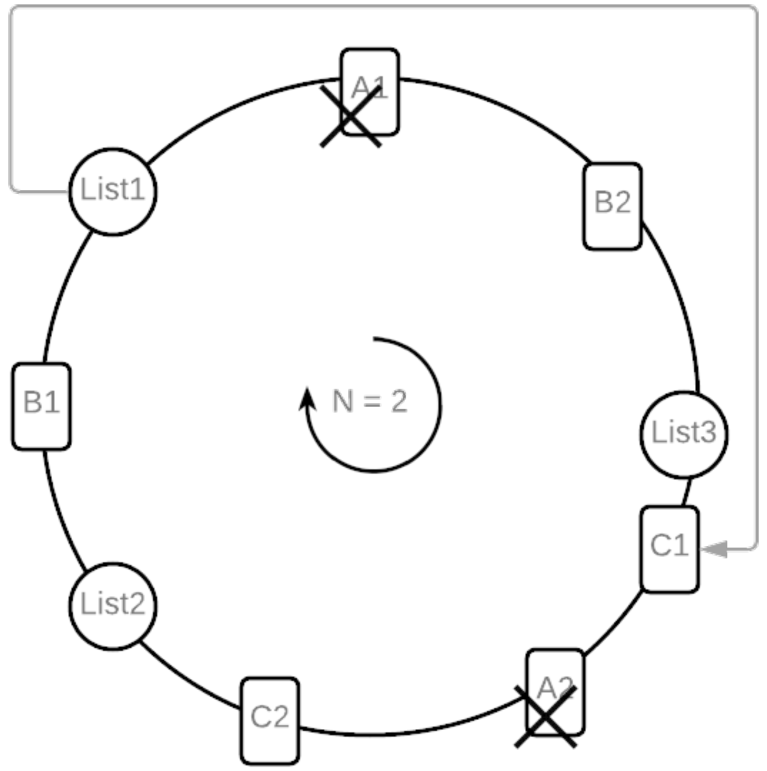
CRDTS IMPLEMENTATION

- For this projects we combined all of the CRDT preciously presented.
- Add-Wins Set: This CRDT was used to keep track of the presence of a product in a shopping list.
- G-Counter: Used to represent the quantity of an item that is already bought (*quantityBought*). This number can only increase.
- PN-Counter: Indicates the quantity that should be bought (*quantity*) of a product in the shopping list. Can be decreased as well as increased.

DATA PARTITIONING

- Used **Consistent Hashing** to distribute lists among servers
- All servers have full knowledge of the server grid
- Usage of a circular hash space with configurable size
- Node changes don't require a complete re-hashing
- Virtual Nodes are used to improve **load balancing**





FAULT TOLERANCE

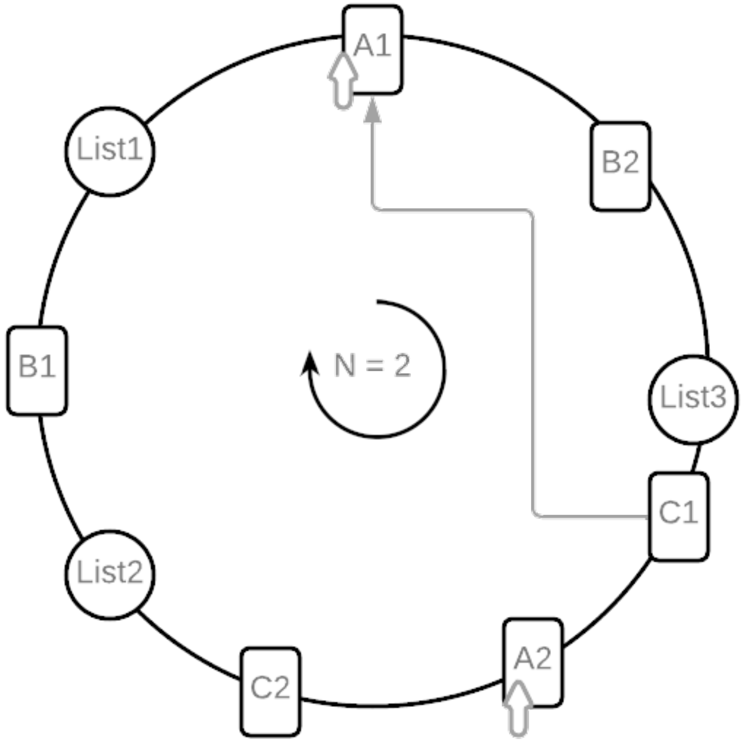
- Because the grid status is fully known, nodes continuously observe others status
- When a node doesn't receive an expected response for a certain interval, a fault is **detected**
- The node that first finds the fault is responsible to broadcast it
- **Hinted Handoff:** After a status change, nodes relocate lists that belong to the sick node, to ensure **replication**
- The image shows the failure of node A and subsequent relocation of List1 to the next available physical node (C)

NODE RECOVERY

- Upon receiving a hinted handoff, the node keeps track which lists are only temporarily relocated
- When a sick node comes back online, all hinted lists are returned, and changes are updated

For **new nodes** joining the network, the process is analogous:

- The new node sends a heartbeat to all the other nodes
- Upon detecting the new node, the servers send it the lists that now belong to it



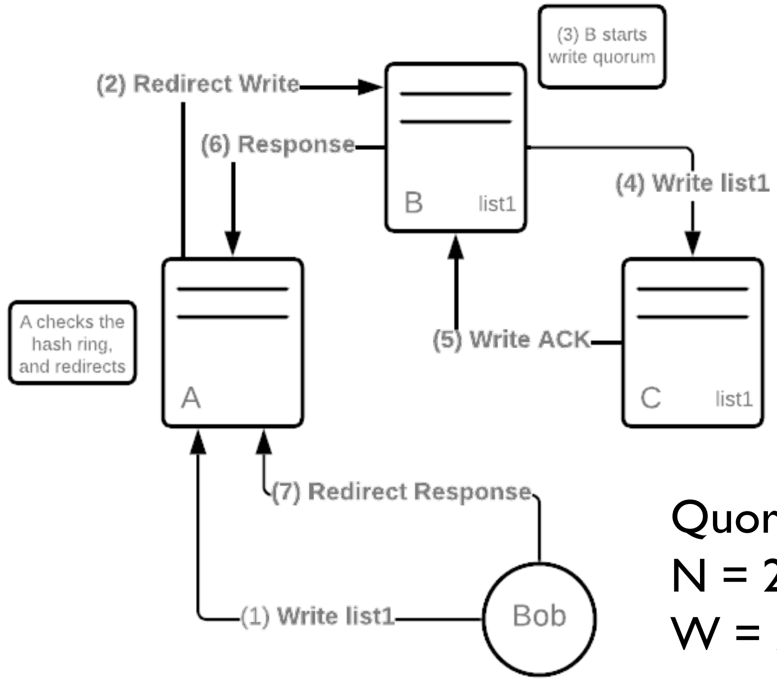
IMPLEMENTATION

- The program was built in **Java**
- Communication is done with ZMQ sockets
- There are two types of processes: **nodes** (servers) and **clients**
- Client-Server communication is done DEALER-ROUTER
    - Client is able to fairly distribute requests among known nodes
    - Servers can talk to multiple clients at the same time
- Server-Server communication is done ROUTER-ROUTER
    - This allows for a star topology and fully interconnected nodes

PROGRAM EXECUTION

LIST WRITE OPERATION

1. Client sends request in a **round-robin** fashion to each server
2. When the server receives a request, this may be redirected, when the node is out of the ring
3. The operation begins by starting a quorum.
4. Node B tells C to write the list
5. Upon receiving confirmation from C, quorum is achieved
6. The response is redirected to A
7. A now sends the reply to the client



Quorum:
N = 2
W = 2

PARAMETER TUNING

The program is built to allow the configuration of parameters, including:

- **Number of virtual nodes**
  - Increasing the number of virtual nodes improves load balancing, by increasing the granularity in distributing data across the hash ring.
  - May have diminishing returns as the hashing complexity increases
- **Ring Size**
  - Ring size can be changed to better suit each application's dimensionality.
  - Increasing this value diminishes the number of collisions but hashes become more sparse
- **Quorum**
  - Varying quorum values N, W and R allows the developer to empower CAP properties best suit to a specific application
  - Higher **N** value enhances partition tolerance by increasing the number of replicas
  - Increasing **R** improves consistent at the cost of availability
- **Timeouts**
  - Configuring delays changes how fast the system reacts to faults

DISCUSSION

- Using a static grid configuration simplifies communications but discourages online modifications. Mechanisms like **dynamic discovery** or the usage of **brokers** can help in solving this problem.
- The system relies on each node having complete network knowledge. This can, however, impose a limit on horizontal scaling as the overhead in maintaining this state increases.
- The focus on availability and the local-first structure gives the user an always ready to use application.
  - The trade-off is a considerable lost in consistency.
  - Using **eventual consistency** reduces user experience in the sense that changes may not be updated as fast as in more online-oriented architechtures

CONCLUSION

- **Scalability:** The product was build with a focus on scalability and allows the addition of multiple servers to the network.
- **Local-First:** An offline-oriented architecture was achieved by prioritizing local actions and updating with the cloud on the background.
- **Collaboration:** By writing operations to the cloud, users are given the option to work on the same data at the same time.
- **Consistency**: The usage of CRDTs empowered the program with the ability to merge data and ensure seamless colaboration.