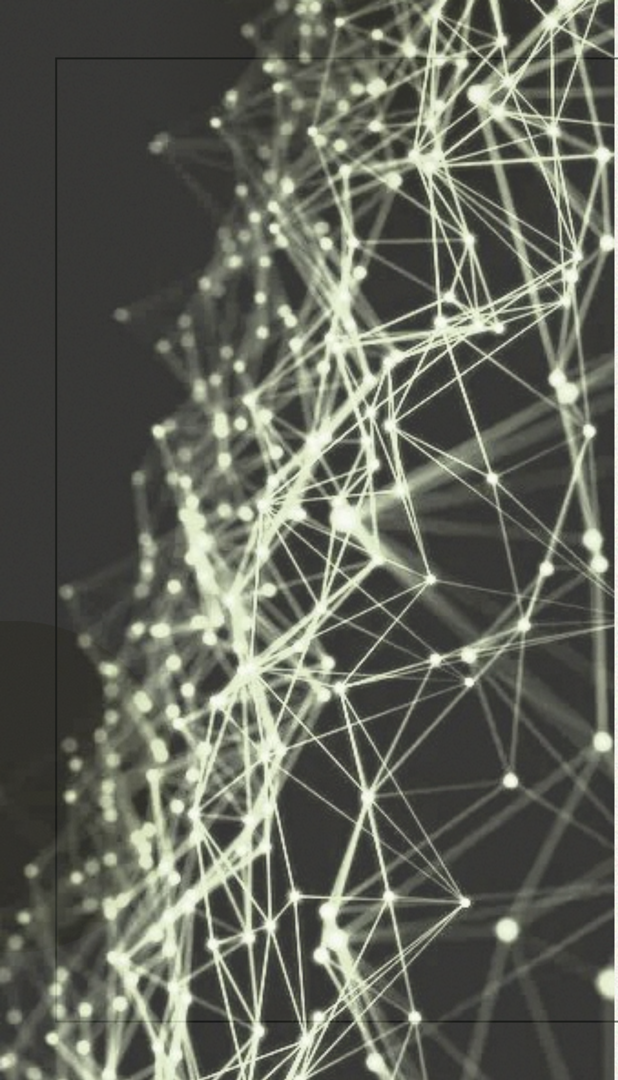
Shopping Lists on The Cloud

Large Scale Distributed Systems

Ricardo Inácio[1] Diogo Lemos[2] Tomás Maciel[3]

FEUP

{up202302742[1], up202003484[2] up202006845[3]}@up.pt



Abstract

- Underlying Context and Functionality
- Local-First Application
- Cloud-Side Architecture Inspired by Amazon Dynamo [1]

[1] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., ... & Vogels, W. (2007). Dynamo: Amazon's highly available key-value store. ACM SIGOPS operating systems review, 41(6), 205-220.

Table of contents

01

Communication Process

Communication Process

Proxy

Servers

Clients

```python
import zmq
import uuid
import time
import pickle
import sys
import os
import json

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), "..")))
from list_writer.LWW.lww import ShoppingList


class Server:
    def __init__(
        self,
        node_type,
        proxy_address_reach,
        proxy_address_online,
        proxy_address_s,
        proxy_address_r,
        proxy_address_ack,
        proxy_address_hello,
    ):
        self.context = zmq.Context()
        self.socket_s = self.context.socket(zmq.DEALER)
        self.socket_r = self.context.socket(zmq.SUB)
        self.socket_reach = self.context.socket(zmq.SUB)
        self.socket_online = self.context.socket(zmq.DEALER)
        self.socket_ack = self.context.socket(zmq.SUB)
        self.socket_hello = self.context.socket(zmq.DEALER)
        self.uuid = str(uuid.uuid4())
        self.node_type = node_type
        self.isAssigned = False
        self.shopping_lists = {}
        self.proxy_address_s = proxy_address_s
        self.proxy_address_r = proxy_address_r
        self.proxy_address_ack = proxy_address_ack
        self.proxy_address_hello = proxy_address_hello
        self.proxy_address_reach = proxy_address_reach
        self.proxy_address_online = proxy_address_online

        self.poller = zmq.Poller()

        self.colorize_text(f"S> STARTED: {self.uuid}\n--------\n")
```

```python
import zmq
import uuid
import time
from hash_ring.hash_ring import HashRing
import pickle
import os
import sys

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), "..")))
from list_writer.LWW.lww import ShoppingList


class Proxy:
    def __init__(
        self,
        frontend_port_s,
        frontend_port_r,
        backend_port_reach,
        backend_port_online,
        backend_port_r,
        backend_port_s,
        backend_port_hello,
        backend_port_ack,
    ):
        self.context = zmq.Context()

        self.frontend_s = self.context.socket(zmq.PUB)
        self.frontend_r = self.context.socket(zmq.ROUTER)
        self.frontend_s.bind(f"tcp://*:{frontend_port_s}")
        self.frontend_r.bind(f"tcp://*:{frontend_port_r}")

        self.servers = []
        self.serverQueue = []

        self.backend_reach = self.context.socket(zmq.PUB)
        self.backend_reach.bind(f"tcp://*:{backend_port_reach}")
        self.backend_reach.setsockopt(zmq.SNDTIMEO, 1000)
        self.backend_online = self.context.socket(zmq.ROUTER)
        self.backend_online.bind(f"tcp://*:{backend_port_online}")
        self.backend_online.setsockopt(zmq.RCVTIMEO, 1000)

        self.backend_s = self.context.socket(zmq.PUB)
        self.backend_s.setsockopt(zmq.SNDTIMEO, 1000)
        self.backend_s.bind(f"tcp://*:{backend_port_s}")
        self.backend_r = self.context.socket(zmq.ROUTER)
        self.backend_r.setsockopt(zmq.RCVTIMEO, 1000)
        self.backend_r.bind(f"tcp://*:{backend_port_r}")
        self.backend_ack = self.context.socket(zmq.PUB)
        self.backend_ack.bind(f"tcp://*:{backend_port_ack}")
        self.backend_hello = self.context.socket(zmq.ROUTER)
        self.backend_hello.setsockopt(zmq.RCVTIMEO, 1000)
        self.backend_hello.bind(f"tcp://*:{backend_port_hello}")

        self.poller = zmq.Poller()
        self.hash_ring = HashRing()

    def add_server(self, server_id):
        if server_id not in self.servers:
            self.servers.append(server_id)
```

```python
import zmq
import uuid
import time
import random
import sys
import os
import pickle

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), "..")))

from ui import UI


class Client:
    def __init__(self, node_type, proxy_address_s, proxy_address_r):
        self.context = zmq.Context()
        self.socket_s = self.context.socket(zmq.DEALER)
        self.socket_r = self.context.socket(zmq.SUB)
        self.uuid = str(uuid.uuid4())
        self.node_type = node_type
        self.proxy_address_s = proxy_address_s
        self.proxy_address_r = proxy_address_r

        self.colorize_text(f"C> STARTED: {self.uuid}\n--------\n")

    def connect(self):
        self.socket_r.connect(self.proxy_address_r)
        self.socket_r.setsockopt(zmq.RCVTIMEO, 1000)
        self.socket_r.setsockopt_string(zmq.SUBSCRIBE, self.uuid)
        self.socket_s.connect(self.proxy_address_s)
        self.socket_s.setsockopt(zmq.SNDTIMEO, 1000)

    def send_data(self, list, list_id):
        # try to upload to server

        tries = 0
        while tries != 3:
            tries += 1
            try:
                self.socket_s.send_multipart(
                    [self.uuid.encode(), list, list_id.encode()]
                )
                self.colorize_text(f"C> UPLOADING LIST: {list_id}\n")
```

Communication Process

Servers

- Apart from the Hello/ACK, receives and processes the requests made by the client.
- Depending on the request made by the client, either sends the list encoded in bytes (“GET_LIST” request), or decodes the list and save/merge on its storage.
- To keep track of the lists as objects, after an operation made after a request, it instantiates all the lists and saves them on a dictionary (to have correct timestamps)

Proxy

- Works as an intermediary in the communication client-server.
- It contains 8 sockets at total.
- 2 sockets related to the client (1 to handle the responses and other for its requests)
- 6 sockets related to the server:
  - 2 for Hello/ACK
  - 2 to guarantee the presence and availability of the server
  - 2 for the regular communication

Clients

- When the client selects an option in the menu (except 6) and inputs valid parameters, he sends a request with the format [uuid, list, list_id], being the list encoded in bytes (using library *pickle*).
- When he chooses option 6, he sends a request with the same parameters but instead of the list, a string “GET_LIST”.
- After that, it waits for a response of the server

02

Servers

02Servers

Store and **Serve** the latest version of lists

Phases:

1. HELLO / ACK
2. REACH / ONLINE
3. Listening → Send / Merge and Save

poller()

Methods:

- instantiate_lists()
- start_listening()
- get_list()
- save_list_server_to_file()
- get_list_from_storage()

sockets ***attributes***

```
self.socket_s = self.context.socket(zmq. DEALER)
self.socket_r = self.context.socket(zmq. SUB)
self.socket_reach = self.context.socket(zmq. SUB)
self.socket_online = self.context.socket(zmq. DEALER)
self.socket_ack = self.context.socket(zmq. SUB)
self.socket_hello = self.context.socket(zmq. DEALER)
self.uuid = str(uuid.uuid4())
self.node_type = node_type
self.isAssigned = False
self.proxy_address_s = proxy_address_s
self.proxy_address_r = proxy_address_r
self.proxy_address_ack = proxy_address_ack
self.proxy_address_hello = proxy_address_hello
self.proxy_address_reach = proxy_address_reach
self.proxy_address_online = proxy_address_online
self.shopping_lists = {}
```

self.shopping_lists[list_id] = ShoppingList()

```
self.socket_r.connect(self.proxy_address_r)
self.socket_r.setsockopt_string( zmq.SUBSCRIBE, self.uuid)
self.socket_reach.connect(self.proxy_address_reach)
self.socket_reach.setsockopt_string( zmq.SUBSCRIBE, self.uuid)
self.socket_ack.connect(self.proxy_address_ack)
self.socket_ack.setsockopt_string( zmq.SUBSCRIBE, self.uuid)
self.socket_hello.connect(self.proxy_address_hello)
self.socket_online.connect(self.proxy_address_online)
self.socket_s.connect(self.proxy_address_s)
```

connect()

03

Clients

03Clients

Processes behind the **UI** for **Users**

Send:

- 3 tries:
  - Using **try / except** blocks:
  - If raises **exception**, retries.
  - Else, Uploads Successfully.

*Always saves **list** locally **(local-first)***

Phases:

1. Connects
2. Calls UI
3. Send or Request Lists

```
self.socket_s.send_multipart(
[self.uuid.encode(), list, list_id.encode()]
)
```

Get:

- 3 tries:
  - Using **try / except** blocks:
  - If raises **exception**, retries.
  - Else, Fetches Successfully.

```
self.context = zmq.Context()
self.socket_s = self.context.socket(zmq. DEALER)
self.socket_r = self.context.socket(zmq. SUB)
self.uuid = str(uuid.uuid4())
self.node_type = node_type
self.proxy_address_s = proxy_address_s
self.proxy_address_r = proxy_address_r
```

sockets
attributes

```
self.socket_r.connect(self.proxy_address_r)
self.socket_r.setsockopt(zmq. RCVTIMEO, 1000)
self.socket_r.setsockopt_string(zmq. SUBSCRIBE, self.uuid)
self.socket_s.connect(self.proxy_address_s)
self.socket_s.setsockopt(zmq. SNDTIMEO, 1000)
```

connect()

04

Proxy

04 Proxy

The **Middleware** and **Node Manager**

```
def add_server(self, server_id):
        if server_id not in self.servers:
                self.servers.append(server_id)
                self.serverQueue.append(server_id)

                if len(self.serverQueue) == 5:
                self.hash_ring.generate_ring(self.serverQueue)
                self.hash_ring.print_key_ranges(index=True)
                self.serverQueue.clear()
```

```
def remove_server(self, server_id):
        if server_id in self.servers:
                self.servers.remove(server_id)
                self.hash_ring.remove_node(server_id)
```

sockets ***attributes***

```
self.frontend_s = self.context.socket(zmq. PUB)
self.frontend_r = self.context.socket(zmq. ROUTER)
self.frontend_s.bind(f"tcp://*:{frontend_port_s}")
self.frontend_r.bind(f"tcp://*:{frontend_port_r}")
```

```
self.backend_reach = self.context.socket(zmq. PUB)
self.backend_reach.bind(f"tcp://*:{backend_port_reach}")
self.backend_reach.setsockopt( zmq.SNDTIMEO, 1000)
self.backend_online = self.context.socket(zmq. ROUTER)
self.backend_online.bind(f"tcp://*:{backend_port_online}")
self.backend_online.setsockopt( zmq.RCVTIMEO, 1000)
```

```
self.backend_s = self.context.socket(zmq. PUB)
self.backend_s.setsockopt( zmq.SNDTIMEO, 1000)
self.backend_s.bind(f"tcp://*:{backend_port_s}")
self.backend_r = self.context.socket(zmq. ROUTER)
self.backend_r.setsockopt( zmq.RCVTIMEO, 1000)
self.backend_r.bind(f"tcp://*:{backend_port_r}")
self.backend_ack = self.context.socket(zmq. PUB)
self.backend_ack.bind(f"tcp://*:{backend_port_ack}")
self.backend_hello = self.context.socket(zmq. ROUTER)
self.backend_hello.setsockopt( zmq.RCVTIMEO, 1000)
self.backend_hello.bind(f"tcp://*:{backend_port_hello}")
```

```
self.poller = zmq.Poller()          self.servers = []
self.hash_ring = HashRing()         self.serverQueue = []
```

02/02

04Proxy

The **Middleware** and **Node Manager**

*find the **node***

```
def select_responsible_node(self, key):
        self.colorize_text(f"P> NODE
({self.hash_ring.lookup_node(key)}) SELECTED\n")
        return self.hash_ring.lookup_node(key)
```

```
if destination_node:    #(destination = self.hash_ring.lookup_node(list_id))
  self.colorize_text(f"P> SENDING S_REACH: { destination_node}\n")
  try:
   self.backend_reach.send_multipart([ destination_node.encode(),b"S_REACH"])
   s_online = self.backend_online.recv_multipart()

 if s_online[2] == b"S_ONLINE":
    self.colorize_text("P> S_ONLINE RECEIVED\n")
    server_uuid =  destination_node
    try:
     self.backend_s.send_multipart([server_uuid.encode(),   message[2],
client_id.encode(), list_id.encode()])
```

Shopping List

*send to correct **node***

```
if self.backend_r in sockets and self.backend_r. poll(0):
       message = self.backend_r.recv_multipart()

       client_id =  message[1]
        list_id =  message[3]

self.frontend_s.send_multipart([client_id,  message[2],
list_id])
```

Shopping List

*sending to **client***

```
def colorize_text(self, text):
  prefix = text[:3]
  switch = {
   "P> ": "\033[ 95m" + text + "\033[0m",  # Purple
   "C> ": "\033[ 94m" + text + "\033[0m",  # Blue
   "S> ": "\033[ 92m" + text + "\033[0m",  # Green
   "HR>": "\033[ 91m" + text + "\033[0m",  # Red
       }

       colored_text = switch.get(prefix, text)
       print(colored_text)
```

colour the output

```
self.poller.register(self. frontend_r, zmq.POLLIN)
self.poller.register(self. backend_r, zmq.POLLIN)
self.poller.register(self. backend_hello, zmq.POLLIN)
```

05

Lists Schema

Lists Schema

Since in this project we want to provide the dissemination of shopping lists among a network of clients and servers, in a distributed manner, we needed to define a concise and well-organised structure of the lists.

The lists are composed of two main elements, the version number, increased with each operation, and an object array, named “list”.

Following this definition all processes and operations can be completed successfully.

```
{
    "version": "1.0",
    "list": [
        {
            "id": "eggs",
            "acquired": "false",
            "quantity": "4"
        }
    ]
}
```

06

CRDT

06CRDTs

Last Writer Wins

CRDTs (Conflict-Free Replicated Data Type) are fulcral to the integrity of our project as they provide a better way to manage versioning and handling of inconsistencies across elements on the network, regarding the distributed data.

We decided to implement and build upon the concept of **Last Writer Wins** or LWW.

This is an element-set configuration to handle list versioning by assigning **timestamp** to each procedure allowing for a conflictless merge operations.

Merge Operations

```python
def merge(self, other):
        merged_list = ShoppingList()

        for element, details in self.add_set.items():
            if element in other.add_set:
                if details["timestamp"] > other.add_set[element]["timestamp"]:
                    merged_list.add_set[element] = details
                    merged_list.add_set[element]["timestamp"] = time.time()
                else:
                    merged_list.add_set[element] = other.add_set[element]
                    merged_list.add_set[element]["timestamp"] = time.time()

            else:
                merged_list.add_set[element] = details
                merged_list.add_set[element]["timestamp"] = time.time()

        for element, details in other.add_set.items():
            if element not in merged_list.add_set:
                merged_list.add_set[element] = details
                merged_list.add_set[element]["timestamp"] = time.time()

        for element, details in self.remove_set.items():
            if element in other.remove_set:
                if details["timestamp"] > other.remove_set[element]["timestamp"]:
                    merged_list.remove_set[element] = details
                    merged_list.remove_set[element]["timestamp"] = time.time()

                else:
                    merged_list.remove_set[element] = other.remove_set[element]
                    merged_list.remove_set[element]["timestamp"] = time.time()

            else:
                merged_list.remove_set[element] = details
                merged_list.remove_set[element]["timestamp"] = time.time()

        for element, details in other.remove_set.items():
            if element not in merged_list.remove_set:
                merged_list.remove_set[element] = details
                merged_list.remove_set[element]["timestamp"] = time.time()

        return merged_list
```

06CRDTs

Last Writer Wins

Add

```python
def add(self, item):
        item_id = item["id"]
        quantity = item.get("quantity", 1)
        acquired = item.get("acquired", "false")

        if (
            item_id not in self.add_set
            or self.add_set[item_id]["timestamp"] < time.time()
        ):
            self.add_set[item_id] = {
                "timestamp": time.time(),
                "quantity": quantity,
                "acquired": acquired,
            }
        else:
            self.add_set[item_id]["quantity"] += quantity
            self.add_set[item_id]["acquired"] = acquired
```

Remove

```python
def remove(self, item_id):
        current_time = time.time()
        while True:
            try:
                if item_id in self.add_set and (
                    item_id not in self.remove_set
                    or self.remove_set[item_id]["timestamp"] < current_time
                ):
                    self.remove_set[item_id] = {
                        "timestamp": self.add_set[item_id]["timestamp"],
                        "quantity": self.add_set[item_id]["quantity"],
                        "acquired": self.add_set[item_id]["acquired"],
                    }
                    break
                else:
                    raise TypeError("\nItem not in list")
            except TypeError as error:
                print(str(error))
                item_id = input("Please enter a valid item_id: ")
```

06CRDTs

Last Writer Wins

Look Up

```python
def lookup(self, element):
        if element not in self.add_set:
            return False

        if element not in self.remove_set:
            return True

        if self.remove_set[element]["timestamp"] < self.add_set[element]["timestamp"]:
            return True

        return False
```

Acquire

```python
def acquire(self, item_id):
        current_time = time.time()
        while True:
            try:
                if item_id in self.add_set and (
                    item_id not in self.remove_set
                    or self.remove_set[item_id]["timestamp"] < current_time
                ):
                    if self.add_set[item_id]["quantity"] == "1":
                        self.add_set[item_id]["quantity"] = "0"
                        self.add_set[item_id]["acquired"] = "true"
                        self.add_set[item_id]["timestamp"] = time.time()

                    elif self.add_set[item_id]["quantity"] == "0":
                        raise TypeError("\nItem already acquired")
                    else:
                        self.add_set[item_id]["quantity"] = str(
                            int(self.add_set[item_id]["quantity"]) - 1
                        )
                        self.add_set[item_id]["timestamp"] = time.time()

                    break
                else:
                    raise TypeError("\nItem not in list")
            except TypeError as error:
                print(str(error))
                item_id = input("Please enter a valid item_id: ")
```

07

Hash Ring

Merge Operations

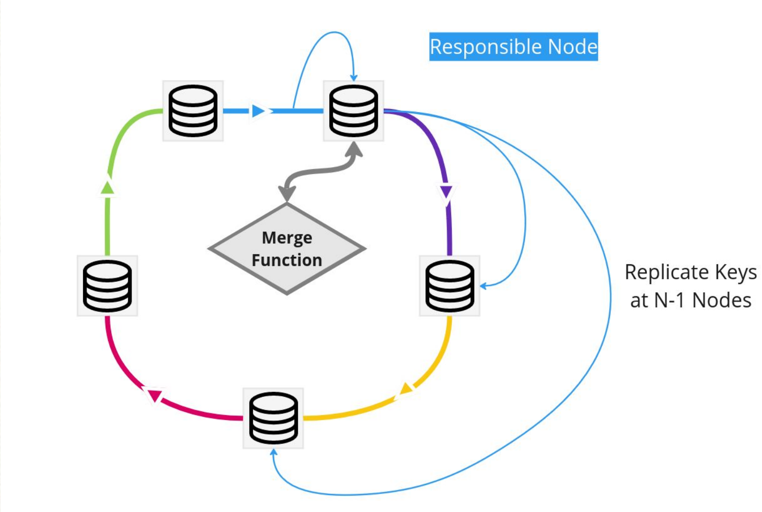
07Hash Ring

Efficient Key Distribution

For this project we recurred to the usage of an hash ring. This allows for adaptation as the data grows bigger and bigger.

By using **virtual nodes**, these represent logical positions on the ring where data items are assigned based on hashed keys.

In our project the service starts with **five** nodes, with **three** virtual nodes each, meaning there are **15** different positions data items can land initially



07Hash Ring

Main Functions

_hash

Using python module **hashlib** we use the md5 for the key hash which then represents the positions of data and nodes on the ring, allowing for an uniform distribution

Lookup node

```
def lookup_node(self, key):
        if not self.ring:
            return None

        hash_key = self._hash(key)

        for ring_key in self.sorted_keys:
            if hash_key < ring_key:
                return self.ring[ring_key]

        return self.ring[self.sorted_keys[0]]
```

07Hash Ring

Main Functions

Add node

```python
def add_node(self, node):
        for i in range(self.virtual_nodes):
            hash_key = self._hash(f"{node}-{i+1}")
            self.ring[hash_key] = node
            self.colorize_text(f"HR> ADDED {node} [VN {i+1}] ({hash_key})\n")
            self.sorted_keys.append(hash_key)
```

Remove Node

```python
def remove_node(self, node):
        keys_to_remove = []
        for i in range(self.virtual_nodes):
            hash_key = self._hash(f"{node}-{i}")
            if hash_key in self.ring:
                keys_to_remove.append(hash_key)
                self.colorize_text(f"HR> REMOVED {node} [VN {i+1}] ({hash_key})\n")

        next_node_index = (self.sorted_keys.index(keys_to_remove[-1]) + 1) % len(
            self.sorted_keys
        )
        next_node = self.sorted_keys[next_node_index]

        source_location = f"server_{node}"
        destination_location = f"server_{next_node}"
        if os.path.exists(source_location):
            os.makedirs(destination_location, exist_ok=True)
            files = os.listdir(source_location)
            for file in files:
                source_path = os.path.join(source_location, file)
                destination_path = os.path.join(destination_location, file)
                shutil.move(source_path, destination_path)

        for key in keys_to_remove:
            self.sorted_keys.remove(key)

        new_ring = {}
        for key in self.sorted_keys:
            new_ring[key] = self.ring[key]

        self.ring = new_ring
```

07Hash Ring

Other functions implemented

Get Replicas

It is very similar to the lookup Node, but instead it returns the previous **N** nodes on the ring.
This allows for the **replication** described previously.

get responsible nodes

Simply informational.
It returns a dictionary with all nodes, and the respective ranges in keyspace.

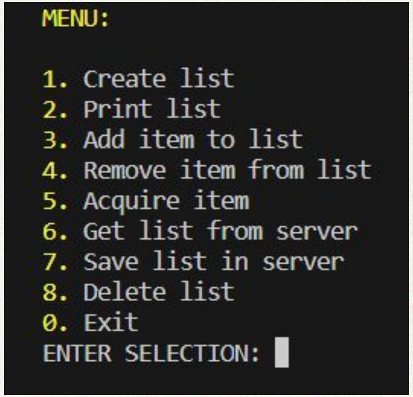
print key ranges

This method is a way to visualize with added detail the ranges of keys for each active node in the system.
It shows how many nodes currently exist in the ring, and for each, a list of each of the key ranges.
Since the keys follow the notation of **md5** we decided to include an option named **index** to provide better understanding of the node allocations and to show the **uniformity** of the distribution itself.

08

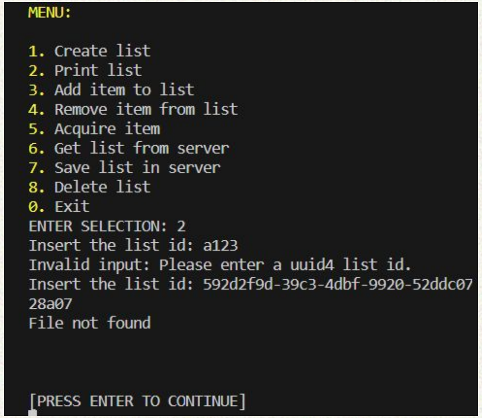
User Interface

User Interface



This is the interface that is available to the client when he initiates the system. He is able to choose any functionality present on the menu, so that he can create and modify shopping lists.

It has a minimalistic design and is implemented on the command line, since the design aspect was never what we valued more in this project.



For each choice (except 1), it’s made a verification to assure that the list id provided is valid. First, we make a semantic verification, to check if it is in uuid format, then we check if the client has a list with that id in its storage.

1. Create list

```
ENTER SELECTION: 1
Insert the item name:pao
Insert the item quantity: 2

Your list:

ITEM {'id': 'pao', 'acquired': 'false', 'quantity':
 '2'}

List saved as /mnt/d/Faculdade/4Y/SDLE/SDLE-2324-TP
1/storage/client_58a96847-a682-477e-9c75-57706dd7f1
52/list_40cd513e-ba88-4109-a63d-ec048949125f.json
C> UPLOADING LIST: 40cd513e-ba88-4109-a63d-ec048949
125f

C> [TRIES: 1]

C> UPLOAD SUCCESSFUL: MERGED IN SERVER

C> LIST SAVED LOCALLY



[PRESS ENTER TO CONTINUE]
```

3. Add item to list

```
ENTER SELECTION: 3
Insert the list id: 40cd513e-ba88-4109-a63d-ec0489
49125f
Insert the item name:agua
Insert the item quantity: 1
List loaded from /mnt/d/Faculdade/4Y/SDLE/SDLE-232
4-TP1/storage/client_5cd93af4-d839-4f00-9b4d-75be9
4516b58/list_40cd513e-ba88-4109-a63d-ec048949125f.
json


Your list:

ITEM {'id': 'pao', 'acquired': 'false', 'quantity'
: '2'}
ITEM {'id': 'agua', 'acquired': 'false', 'quantity
': '1'}

List saved as /mnt/d/Faculdade/4Y/SDLE/SDLE-2324-T
P1/storage/client_5cd93af4-d839-4f00-9b4d-75be9451
6b58/list_40cd513e-ba88-4109-a63d-ec048949125f.jso
n
C> UPLOADING LIST: 40cd513e-ba88-4109-a63d-ec04894
9125f

C> [TRIES: 1]

C> UPLOAD SUCCESSFUL: MERGED IN SERVER

C> LIST SAVED LOCALLY



[PRESS ENTER TO CONTINUE]
```

2. Print list

```
ENTER SELECTION: 2
Insert the list id: 40cd513e-ba88-4109-a63d-ec04894
9125f
List loaded from /mnt/d/Faculdade/4Y/SDLE/SDLE-2324
-TP1/storage/client_c948dca7-ac2b-4260-90e9-007826b
7706b/list_40cd513e-ba88-4109-a63d-ec048949125f.jso
n


Your list:

ITEM {'id': 'pao', 'acquired': 'false', 'quantity':
 '2'}
ITEM {'id': 'agua', 'acquired': 'false', 'quantity'
: '1'}


[PRESS ENTER TO CONTINUE]
```

4. Remove item from list

```
ENTER SELECTION: 4
Insert the list id: 40cd513e-ba88-4109-a63d-ec0489
49125f
Insert the item name:pao
shopping_list a197ddb1-5c1a-45c2-a4f5-d35332c9e248
List loaded from /mnt/d/Faculdade/4Y/SDLE/SDLE-232
4-TP1/storage/client_09379374-bcf2-4656-b245-71e6a
d8d8e6e/list_40cd513e-ba88-4109-a63d-ec048949125f.
json


Your list:

ITEM {'id': 'agua', 'acquired': 'false', 'quantity
': '1'}

List saved as /mnt/d/Faculdade/4Y/SDLE/SDLE-2324-T
P1/storage/client_09379374-bcf2-4656-b245-71e6ad8d
8e6e/list_40cd513e-ba88-4109-a63d-ec048949125f.jso
n
C> UPLOADING LIST: 40cd513e-ba88-4109-a63d-ec04894
9125f

C> [TRIES: 1]

C> UPLOAD SUCCESSFUL: MERGED IN SERVER

C> LIST SAVED LOCALLY



[PRESS ENTER TO CONTINUE]
```

5. Acquire item

```
ENTER SELECTION: 5
Insert the list id: 40cd513e-ba88-4109-a63d-ec04894
9125f
Insert the item name:agua
List loaded from /mnt/d/Faculdade/4Y/SDLE/SDLE-2324
-TP1/storage/client_c948dca7-ac2b-4260-90e9-007826b
7706b/list_40cd513e-ba88-4109-a63d-ec048949125f.jso
n


Your list:

ITEM {'id': 'agua', 'acquired': 'true', 'quantity':
 '0'}

List saved as /mnt/d/Faculdade/4Y/SDLE/SDLE-2324-TP
1/storage/client_c948dca7-ac2b-4260-90e9-007826b770
6b/list_40cd513e-ba88-4109-a63d-ec048949125f.json
C> UPLOADING LIST: 40cd513e-ba88-4109-a63d-ec048949
125f

C> [TRIES: 1]

C> UPLOAD SUCCESSFUL: MERGED IN SERVER

C> LIST SAVED LOCALLY



[PRESS ENTER TO CONTINUE]
```

6. Get list from server

```
ENTER SELECTION: 6
Insert the list id: 40cd513e-ba88-4109-a63d-ec04894
9125f
C> WAITING FOR LIST...

C> [TRIES: 1]


List saved as /mnt/d/Faculdade/4Y/SDLE/SDLE-2324-TP
1/storage/client_c948dca7-ac2b-4260-90e9-007826b770
6b/list_40cd513e-ba88-4109-a63d-ec048949125f.json

[PRESS ENTER TO CONTINUE]
```

7. Save list in server

```
ENTER SELECTION: 7
Insert the list id: 40cd513e-ba88-4109-a63d-ec04894
9125f
List loaded from /mnt/d/Faculdade/4Y/SDLE/SDLE-2324
-TP1/storage/client_c948dca7-ac2b-4260-90e9-007826b
7706b/list_40cd513e-ba88-4109-a63d-ec048949125f.jso
n

C> UPLOADING LIST: 40cd513e-ba88-4109-a63d-ec048949
125f

C> [TRIES: 1]

C> UPLOAD SUCCESSFUL: MERGED IN SERVER

C> LIST SAVED LOCALLY


[PRESS ENTER TO CONTINUE]
```

8. Delete list

```
ENTER SELECTION: 8
Insert the list id: 40cd513e-ba88-4109-a63d-ec0489
49125f
List loaded from /mnt/d/Faculdade/4Y/SDLE/SDLE-232
4-TP1/storage/client_09379374-bcf2-4656-b245-71e6a
d8d8e6e/list_40cd513e-ba88-4109-a63d-ec048949125f.
json

List deleted

[PRESS ENTER TO CONTINUE]
```

0. Exit

```
ENTER SELECTION: 0
EXITING...
```

Conclusions

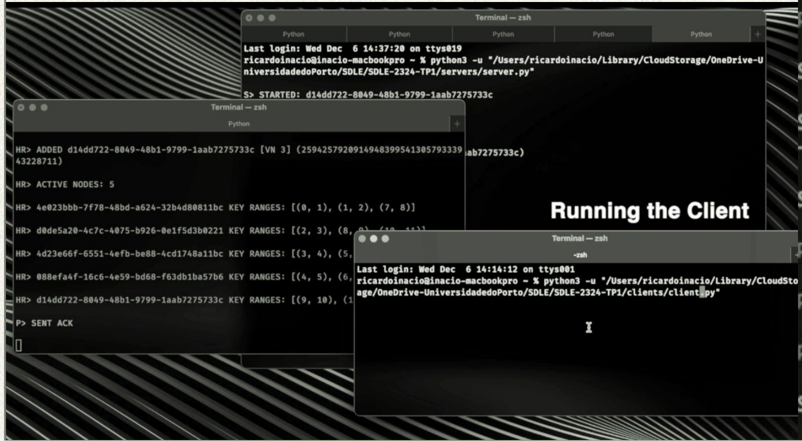
In conclusion, our project successfully developed a distributed system that efficiently manages shopping lists. This innovative system empowers clients with the ability to create, modify, and manage their shopping lists easily. Key functionalities include adding, removing, and marking items as acquired, alongside the capability to save these lists on a server for persistent storage and even get there from the servers to save them locally. Users can also delete the lists locally when they intend to, ensuring flexibility and control over their data.

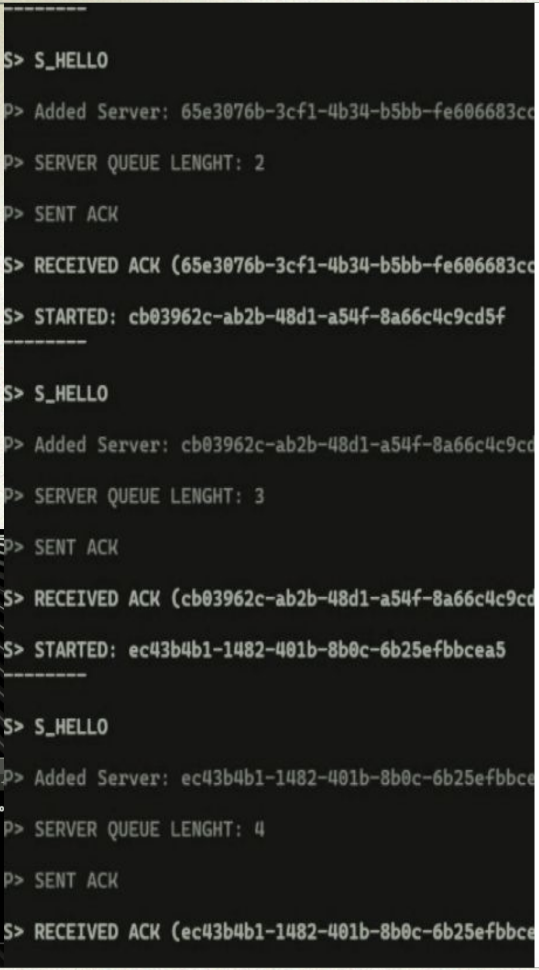
The communication flow client-proxy-server works perfectly fine, which proves the robustness of the system design we developed. In the end we believe that we developed a strong, robust and capable system to ensure the management of shopping lists, which was our goal since the beginning, and with that we also developed our knowledge regarding this type of system.

The End

Ricardo Inácio[1] Diogo Lemos[2] Tomás Maciel[3]

FEUP
{up202302742[1], up202003484[2] up202006845[3]}@up.pt





Shopping Lists on The Cloud

Large Scale Distributed Systems

Ricardo Inácio[1] Diogo Lemos[2] Tomás Maciel[3]

FEUP
{up202302742[1], up202003484[2] up202006845[3]}@up.pt

Abstract

Distributed solutions over the internet can be very complex, due to the sheer amount of mechanisms they should implement, including the support of concurrent control of data, replication, and fault tolerance. In this paper, the group presents a simplistic approach for a large-scale distributed shopping list. The system aims for a local-first[5] implementation, where data should always be accessible and persistent on the client's machine, and the application should run independent of connectivity status. It allows for concurrent read/write operations, with scalability always in consideration, in order to avoid data access bottlenecks. All operations are done on JSON objects, that constitute the shopping lists in question, identifiable by unique ID's (uuid4). They are shared across a network of clients and servers, that must always reach a middleware proxy, while using the **ZeroMQ** messaging library [2] for communication. To handle versioning, the service uses an implementation of a **Last-Writer-Wins CRDT** [4], which supports **add**, **remove**, **compare** and **lookup** functions, as well as **merging**, needed to resolve conflicts. The distributions are managed by a **consistent hash ring**, where all available nodes are uniformly mapped in a topological ring, being each responsible for storing and replicating, a list of data (keys),

Keywords: collaborative, distributed, replication, concurrency, CRDT, LWW, local-first, hash ring.

1 Communication Process

To be able to label this application as collaborative, it must be capable to reach other entities in the network, to transmit a client's lists, as its respective modifications, and to receive new ones, or simply the changes to local outdated versions. All this is achievable by defining a **communication process** that is built on top of a specific **protocol**, that must be followed by all participants.

In the presented service, there are three different entities that participate in the process: the *servers*, which store their designated data (keys) elements, the *clients*, which are able to *create* and *modify* (by adding, removing and checking elements on) shopping lists, and the *proxy*, which acts as a coordinator, by storing the connected servers, handling uniform data distribution by using a **consistent hash ring**, and making sure the correct entities receive the desired data.

The process starts when the ***service.py*** script is executed, and creates an instance of the **Proxy** class, as well as five independent instances of the **Server** class. Each of the latter is programmed to automatically send a "*S_HELLO*" message to the proxy's *socket_hello* (zmq.DEALER) socket until it's assigned to a hash ring position, which is confirmed with an "ACK" message on the *socket_ack* (zmq.SUB) socket by the **Proxy** instance. Then, each server starts listening on the *socket_r* (zmq.SUB) socket for incoming lists, so that it can perform its role-specific operations. At the same time, the server polls this socket alongside the *socket_reach* (zmq.SUB) for **S_REACH** messages, so that it can confirm to the proxy that it is active, and able to communicate, replying with an **S_ONLINE** message through the *socket_online* (zmq.DEALER). It should be noted that each of the *SUB* sockets are given as an option the server's ID, so that only messages specifically meant to arrive to the instance are delivered. The choice to create different sockets for the hello/ack and constant listening processes stemmed from the occurrences where a server started receiving messages/lists from clients, before being assigned a position in the hash ring, since it hadn't received its "*ACK*" message, blocking the service indefinitely.

Afterwards, the service is ready, and able to receive requests, messages, and lists from **Clients**, while also being always listening for new servers that may eventually join the ring, and verifying if the current active ones didn't somehow go offline, and need to be removed from it, all while managing reallocation process, so that data remains consistent and available.