Shopping Lists on the Cloud

Large Scale Distributed Systems - MEIC

2023/2024 - First semester

Group 10

André Filipe Cardoso Barbosa - up202007398

Guilherme Cunha Seco Fernandes de Almeida - up202008866

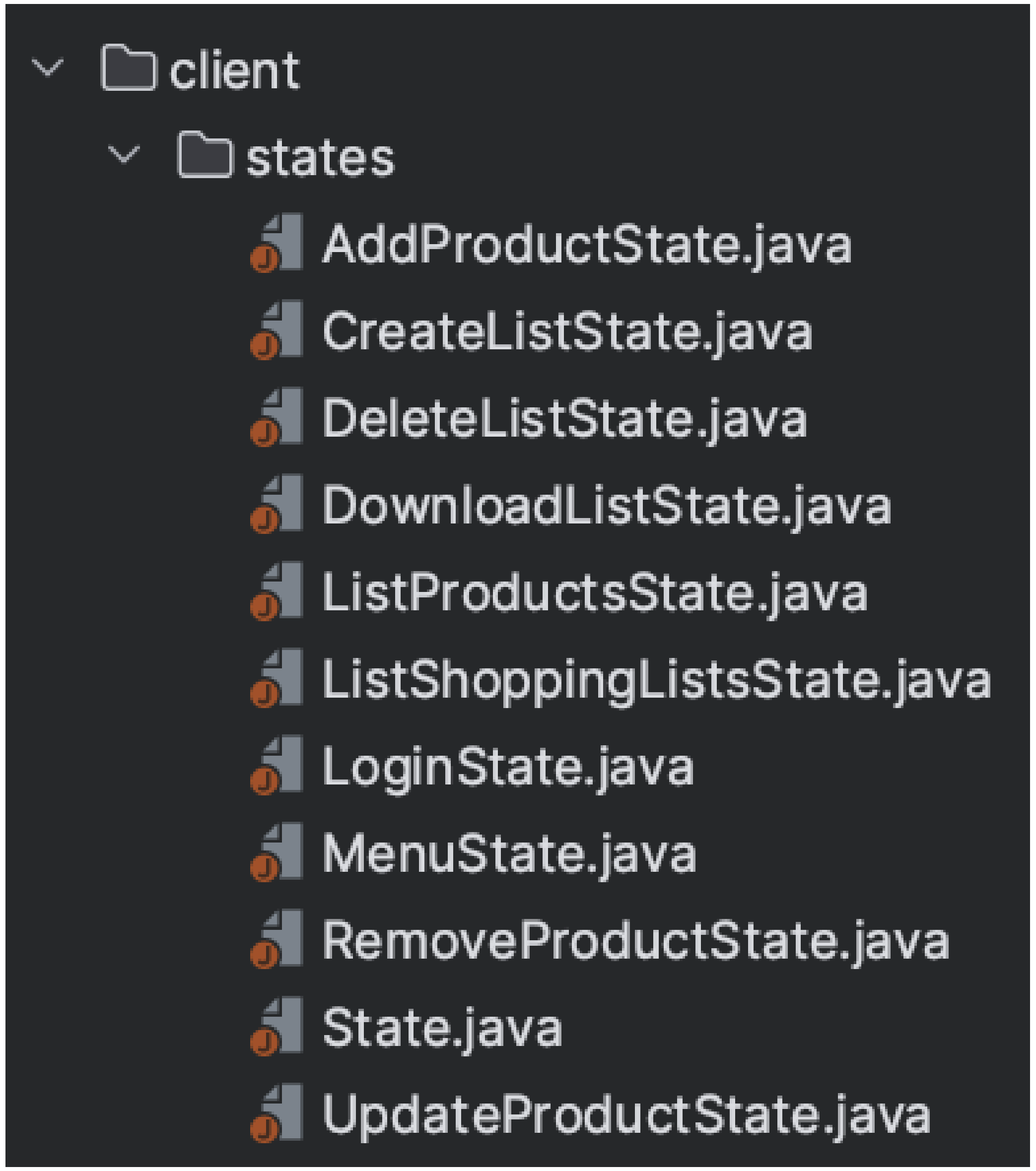
Tiago Filipe Magalhães Barbosa - up202004926

Requirements

- Local-first Application
- Unique ID lists sharing
- CRUD operations over a list
- CRDT based lists
- Data Sharding

Architectural Components - Client

- State Based Program
- CLI Interface
- A Sqlite Database for each user
- Local first approach



```
client
  states
    AddProductState.java
    CreateListState.java
    DeleteListState.java
    DownloadListState.java
    ListProductsState.java
    ListShoppingListsState.java
    LoginState.java
    MenuState.java
    RemoveProductState.java
    State.java
    UpdateProductState.java
```

Local First Implementation

- Server independent
- All CRUD operations can be executed without server connection
- Cloud synchronization performed every N seconds when list opened
- If connection is not established, client ignores and keeps local data

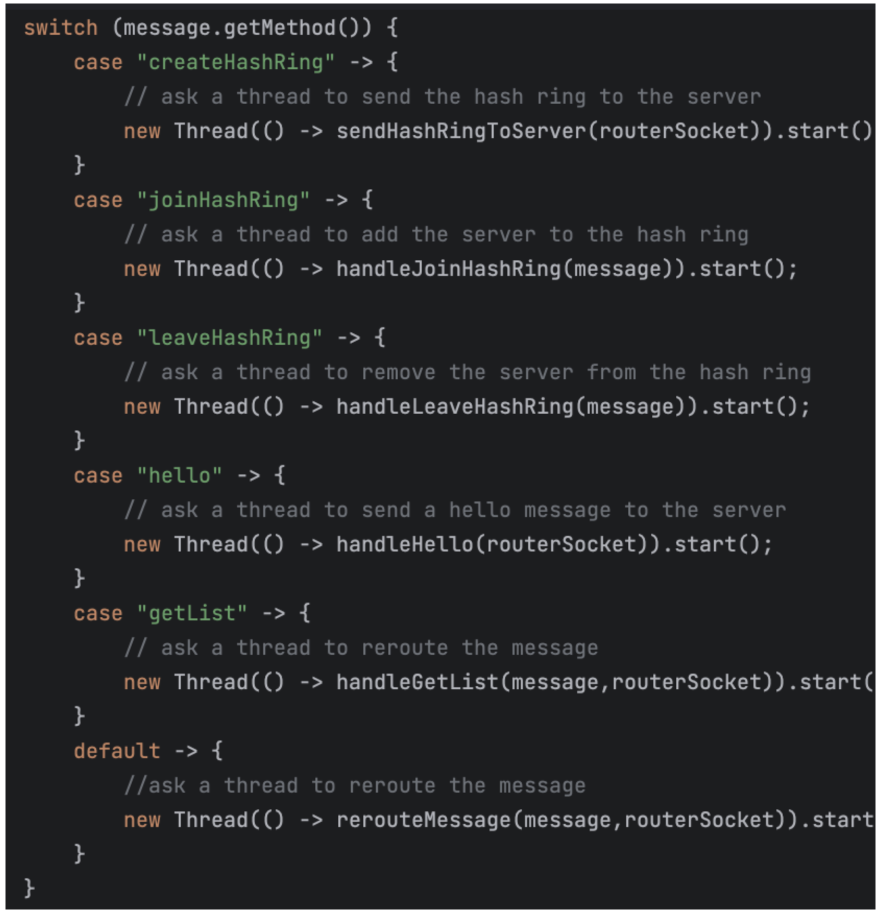
Architectural Components - Router

- Implemented over ZMQ
- Bridge between client and server
- Fixed number of 3 routers
- **All** clients and servers know **all** routers
- Handles hashring

Hash Ring Implementation

- Router
  computes **hashring** with number of servers and virtual nodes, waits for messages:

  - ***createHashring*** - router sends hashring
  - ***joinHashring*** - router recalculates hashring and sends to all servers
  - ***leaveHashring*** – router recalculates hashring and sends to all servers
  - ***getList*** – router tries to get list from responsible server, if fails tries next server
  - no msgs – router calculates server responsible for list, send msg to server with vNode to store list
    - If no server response, router warns client that connection failed

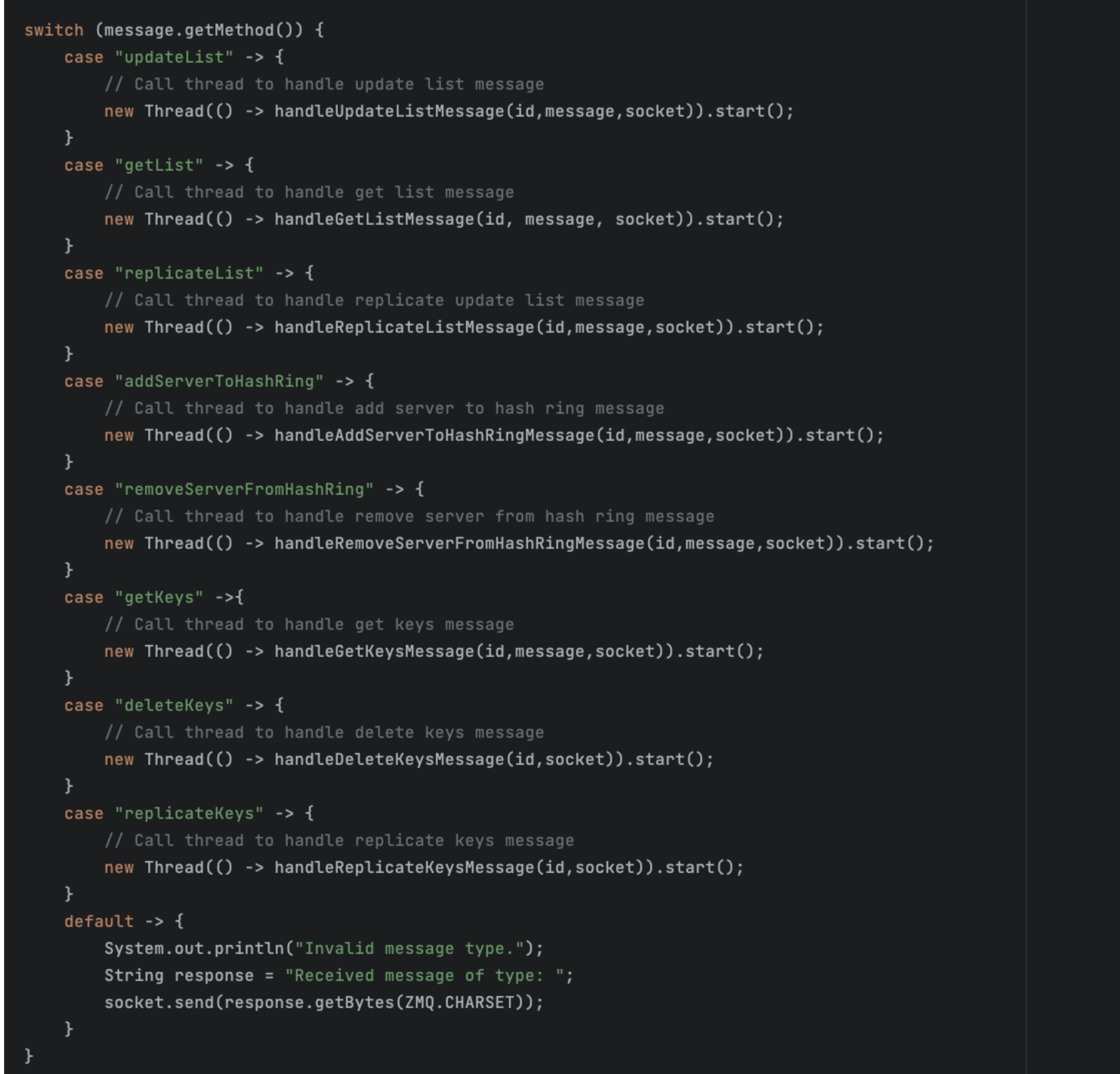


```
switch (message.getMethod()) {
    case "createHashRing" -> {
        // ask a thread to send the hash ring to the server
        new Thread(() -> sendHashRingToServer(routerSocket)).start()
    }
    case "joinHashRing" -> {
        // ask a thread to add the server to the hash ring
        new Thread(() -> handleJoinHashRing(message)).start();
    }
    case "leaveHashRing" -> {
        // ask a thread to remove the server from the hash ring
        new Thread(() -> handleLeaveHashRing(message)).start();
    }
    case "hello" -> {
        // ask a thread to send a hello message to the server
        new Thread(() -> handleHello(routerSocket)).start();
    }
    case "getList" -> {
        // ask a thread to reroute the message
        new Thread(() -> handleGetList(message,routerSocket)).start(
    }
    default -> {
        //ask a thread to reroute the message
        new Thread(() -> rerouteMessage(message,routerSocket)).start
    }
}
```

Architectural Components - Server

- Sqlite database for each server
- All servers know eachother
- Original servers:
  - Connects to router
  - Gets hashring
  - Waits messages
- New thread for each new message
- New server -> New hashring for each server -> Servers check their own keys
- Replication based system
- Hinted Handoff

Server Messages Handling



```
switch (message.getMethod()) {
    case "updateList" -> {
        // Call thread to handle update list message
        new Thread(() -> handleUpdateListMessage(id,message,socket)).start();
    }
    case "getList" -> {
        // Call thread to handle get list message
        new Thread(() -> handleGetListMessage(id, message, socket)).start();
    }
    case "replicateList" -> {
        // Call thread to handle replicate update list message
        new Thread(() -> handleReplicateListMessage(id,message,socket)).start();
    }
    case "addServerToHashRing" -> {
        // Call thread to handle add server to hash ring message
        new Thread(() -> handleAddServerToHashRingMessage(id,message,socket)).start();
    }
    case "removeServerFromHashRing" -> {
        // Call thread to handle remove server from hash ring message
        new Thread(() -> handleRemoveServerFromHashRingMessage(id,message,socket)).start();
    }
    case "getKeys" ->{
        // Call thread to handle get keys message
        new Thread(() -> handleGetKeysMessage(id,message,socket)).start();
    }
    case "deleteKeys" -> {
        // Call thread to handle delete keys message
        new Thread(() -> handleDeleteKeysMessage(id,socket)).start();
    }
    case "replicateKeys" -> {
        // Call thread to handle replicate keys message
        new Thread(() -> handleReplicateKeysMessage(id,socket)).start();
    }
    default -> {
        System.out.println("Invalid message type.");
        String response = "Received message of type: ";
        socket.send(response.getBytes(ZMQ.CHARSET));
    }
}
```

Replication Based System

- Three Layers of storing lists.
- Column on server DB with “replicated” index.

When a new server joins/old server leaves the hashring:

- Keys are redistributed
- Each server recognizes keys that don’t belong to them anymore
- Those keys are marked with 1 on “to_delete” column
- After sending the keys to right server, they are deleted
- All replicas are deleted and replicated again.

Hinted Handoff

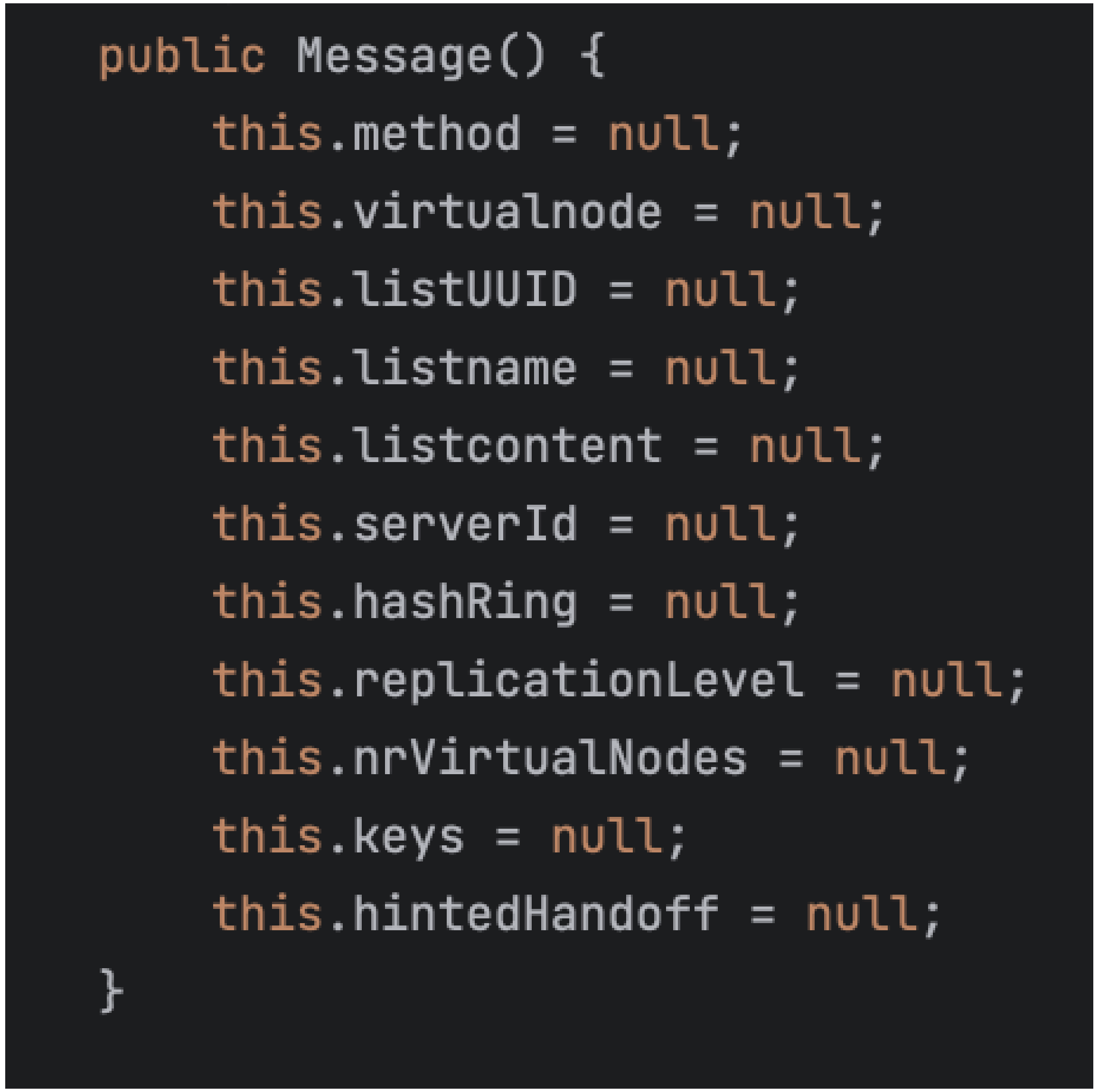
- Occurs when a server that was down comes back.
- Replicas are sent to the next available virtual nodes on hashring.
- Column “hinted_handoff” on DB to store the virtual node to send to.
- Thread periodically verifies if there is non null Hinted Handoff values in each server.

Cross Components – CRDT's

- PNCounter and MapPNCounter classes implemented.
- MapPNCounter refers to a CRDT of a list.
- PNCounter refers to a CRDT of an item from a list.
- Includes value, merge functions.
- User cannot have more negative than positive.
- The whole CRDT is stored in Database.
- ToJson and ToMapCounter functions included.

Cross Components - Message Class

- Message protocol class.
- Involves multiple fields.
- Used on server communication with clients and other servers.
- Transformed in JSON before sent to network.



```
public Message() {
    this.method = null;
    this.virtualnode = null;
    this.listUUID = null;
    this.listname = null;
    this.listcontent = null;
    this.serverId = null;
    this.hashRing = null;
    this.replicationLevel = null;
    this.nrVirtualNodes = null;
    this.keys = null;
    this.hintedHandoff = null;
}
```

Limitations/Improvements

- Hinted handoff
- Current CRDT makes unviable a person removing items that other added
- List is read only if coordenator server is down
- Acquire products

Demo Video