U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Parallel and Distributed Computation

Distributed and Partitioned Key-Value Store

Parallel and Distributed Computing

Degree in Informatics and Computing Engineering

Henrique Pinho
up201805000@up.pt

Luís Marques
up201104354@up.pt

Paulo Ferreira
up201804977@up.pt

2022

Membership Service	Implemented
Storage Service	Implemented
Replication	Not Implemented
Fault-tolerance	Partial Implemented
Concurrency	Thread-Pools Implemented
RMI	Implemented

Membership Service

There is no centralized membership registry, each node must maintain its own view of membership. This is achieved using a gossip-based membership protocol.

JOIN/LEAVE Messages

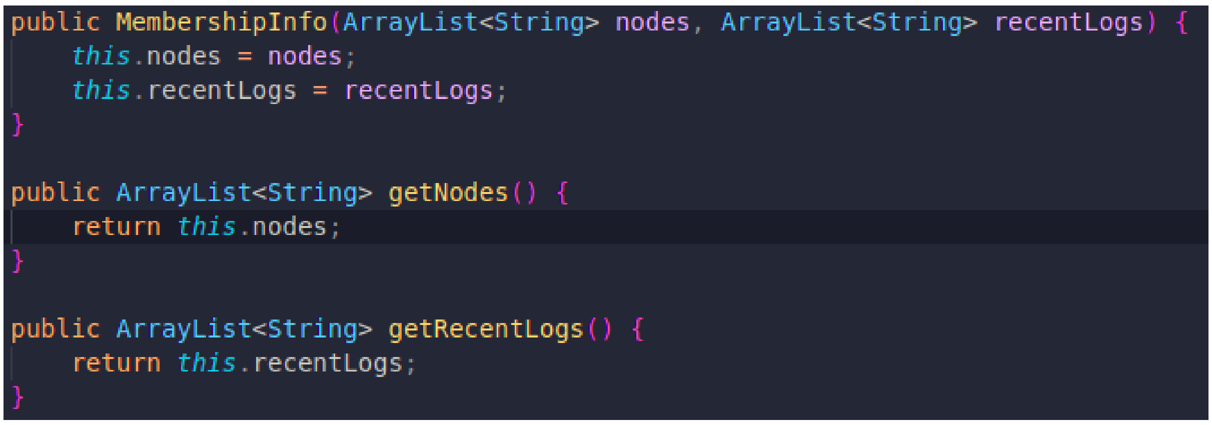
- JOIN <nodeId> <mcastPort> <counter> <CRFL>
- LEAVE <nodeId> <mcastPort> <counter> <CRFL>

Before a node joins the cluster it sends a multicast JOIN message and waits for three other nodes to respond with MembershipInfo through a TCP channel.

If it does not receive the three responses, it sends again the multicast message to a total of three messages.

MEMBERSHIP Messages

- <membershipInfo>
- MEMBERSHIP <CRFL> <membershipInfo>



```java
public MembershipInfo(ArrayList<String> nodes, ArrayList<String> recentLogs) {
    this.nodes = nodes;
    this.recentLogs = recentLogs;
}

public ArrayList<String> getNodes() {
    return this.nodes;
}

public ArrayList<String> getRecentLogs() {
    return this.recentLogs;
}
```

The former format of the membership message is sent through TCP to the recent node that joins the cluster.

The latter format is used in a multicast message sent every second. The java class CastMembershipInfo is used in a scheduledThread.

Initiate Membership Service:

To initiate the Membership Service, three other nodes send through TCP a membershipInfo message.

- membershipInfo: contains a list of all members and the most recent 32 logs

To keep the Membership info almost up to date, the nodes that join the cluster waits 10 seconds before it starts to send membership info to other nodes.

Fault Tolerance:

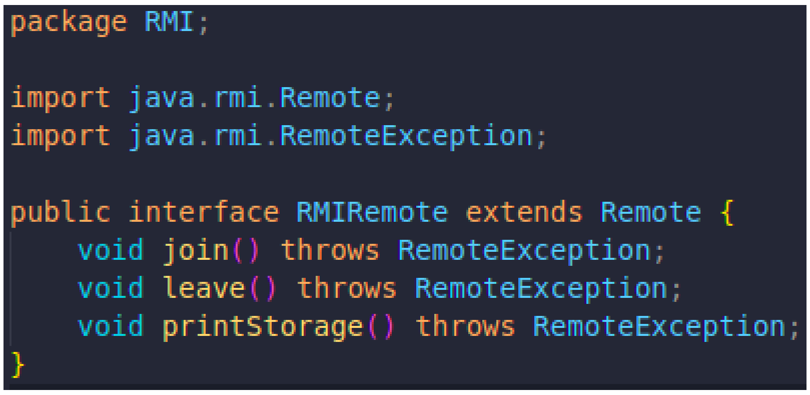
When a node crashes the counter and the log is saved in non volatile memory.

When the node recovers from the crash it loads the last known state.

Each node has a counter that increments by one every time it joins and leaves the cluster. An odd number means that the node is not a member and an even number means that it is a member.

The periodic membership info cast through Multicast is sent as a whole in an object.

RMI:



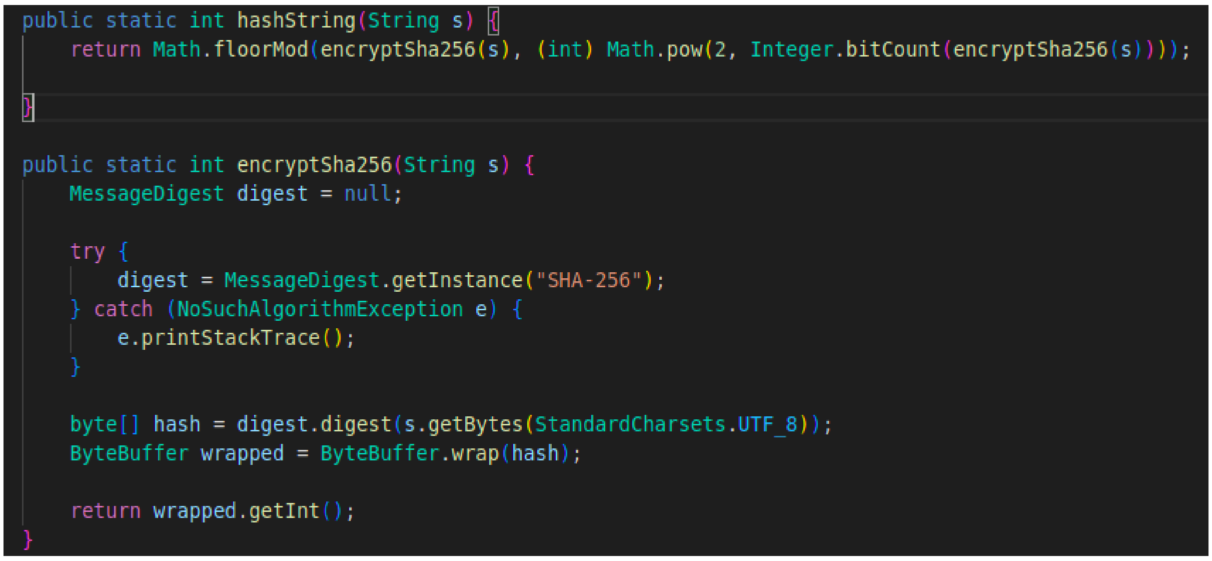
```java
package RMI;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface RMIRemote extends Remote {
    void join() throws RemoteException;
    void leave() throws RemoteException;
    void printStorage() throws RemoteException;
}
```

RMI/RMIRemote.java

Storage Service

Consistent Hashing:

All keys are hashed with a 256-bit hash function which yields its position in the “ring” Nodes are placed randomly in the “ring”. We walk clockwise from the key’s position to find the first node, thus each node is responsible for keys between itself and its predecessor node.

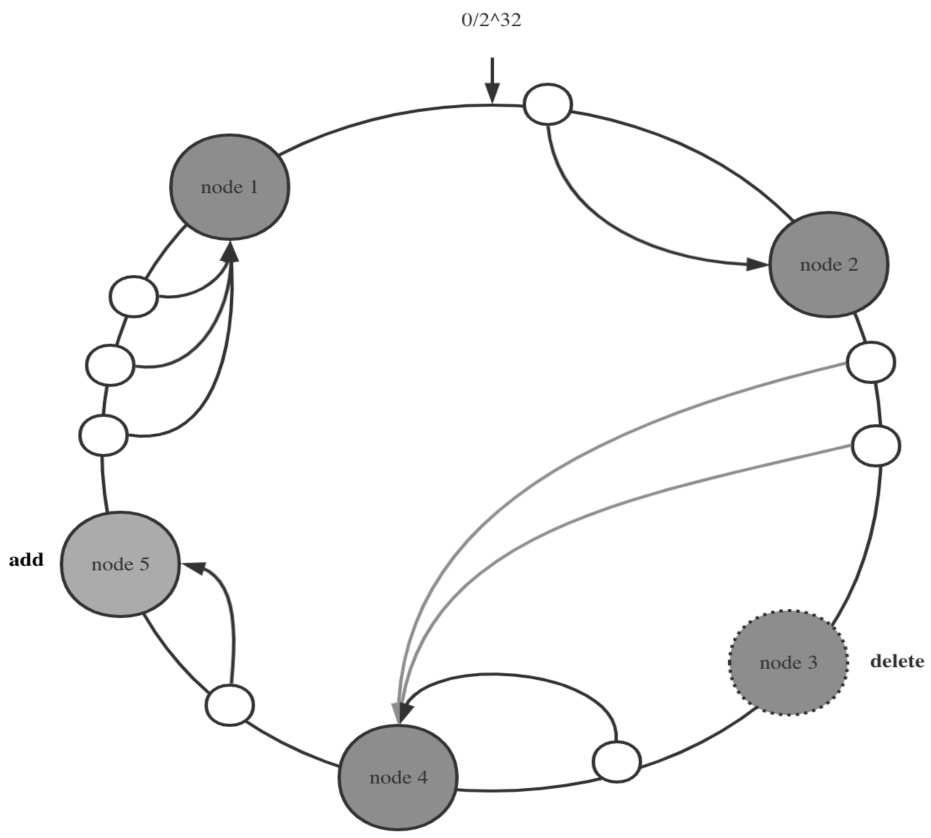


```java
public static int hashString(String s) {
    return Math.floorMod(encryptSha256(s), (int) Math.pow(2, Integer.bitCount(encryptSha256(s))));

}

public static int encryptSha256(String s) {
    MessageDigest digest = null;

    try {
        digest = MessageDigest.getInstance("SHA-256");
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
    }

    byte[] hash = digest.digest(s.getBytes(StandardCharsets.UTF_8));
    ByteBuffer wrapped = ByteBuffer.wrap(hash);

    return wrapped.getInt();
}
```

Hash function

<u>Advantage</u>: An advantage of Consistent Hashing, is that during a resize of the cluster, only a small subsection of items need to be moved to different nodes.

<u>Disadvantage</u>: Random position assignment of nodes leads to non-uniform data and load distribution.



To decide where a piece of data should live in a cluster, Dynamo uses a technique called Consistent Hashing.

The key of the object is run through a hashing function to deterministically derive a number, which is subsequently mapped into a position on a theoretical ring.

The node which should house the information is then determined by finding the next node (which also sits on the ring) in a clockwise direction from that position.

A benefit of Consistent Hashing, is that during a resize of the cluster (adding or removing nodes), only a small subsection of items need to be moved to different nodes.

STORAGE Messages

- PUT <key> <value>
- DELETE <key>
- GET <key>
- <ArrayList key-value>

Searching for the responsible node:

A node, when it receives a PUT/GET/DELETE message, checks the node responsible for that key. If the key belongs to another node it sends through TCP channel a PUT message to the responsible node for that key-value.

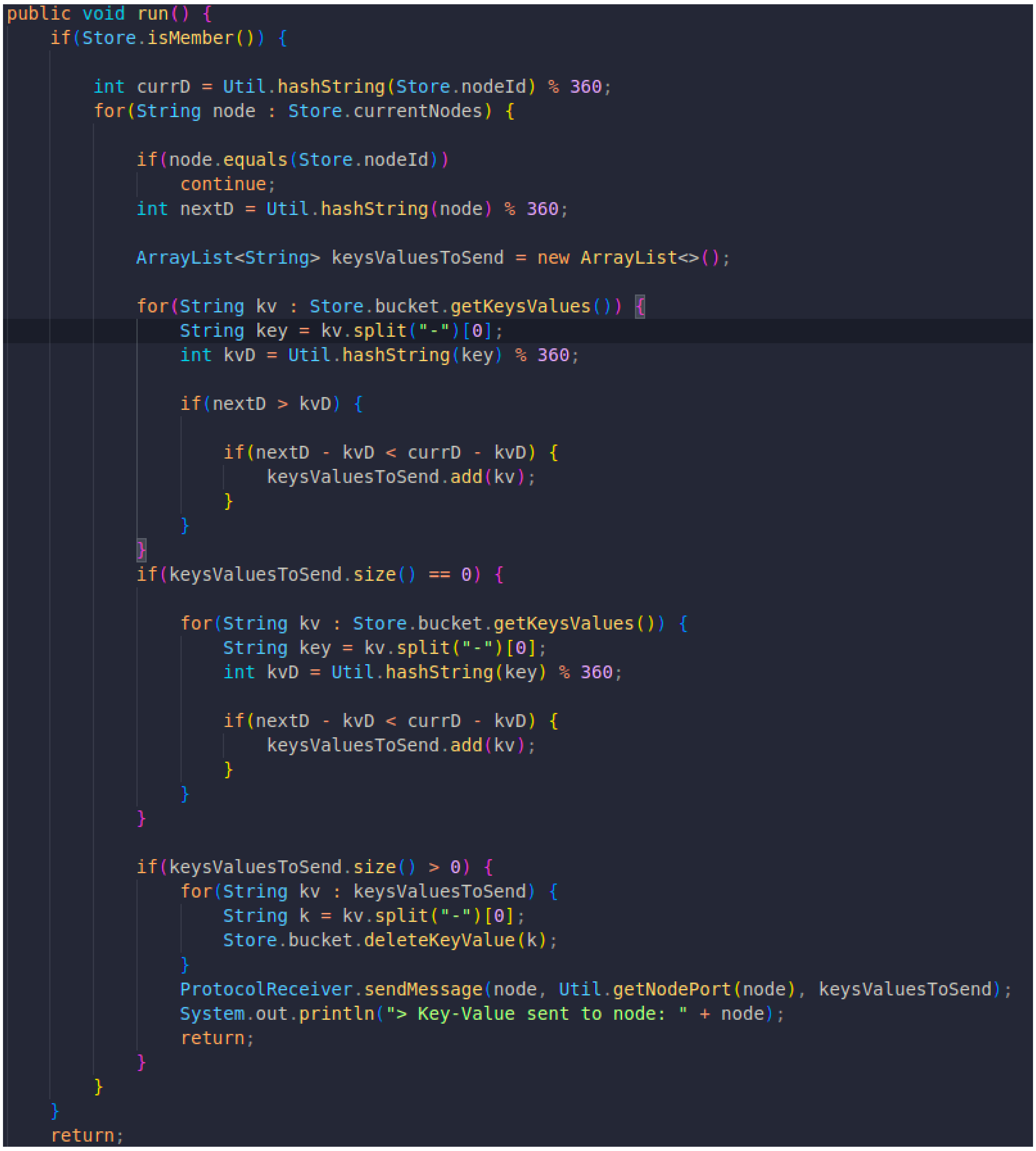
Pair transfer on membership changes:

Before a node leaves the cluster, it sends all the key-value pairs to the successor’s node.

When a node joins the cluster the node’s successor sends the key-value pairs to the recent node.

This exchange of information is done by a transfer key-value message through a TCP channel.

To keep the key-value pairs in the right node a scheduled task is running at a fixed delay.



```java
public void run() {
    if(Store.isMember()) {

        int currD = Util.hashString(Store.nodeId) % 360;
        for(String node : Store.currentNodes) {

            if(node.equals(Store.nodeId))
                continue;
            int nextD = Util.hashString(node) % 360;

            ArrayList<String> keysValuesToSend = new ArrayList<>();

            for(String kv : Store.bucket.getKeysValues()) {
                String key = kv.split("-")[0];
                int kvD = Util.hashString(key) % 360;

                if(nextD > kvD) {

                    if(nextD - kvD < currD - kvD) {
                        keysValuesToSend.add(kv);
                    }
                }
            }
            if(keysValuesToSend.size() == 0) {

                for(String kv : Store.bucket.getKeysValues()) {
                    String key = kv.split("-")[0];
                    int kvD = Util.hashString(key) % 360;

                    if(nextD - kvD < currD - kvD) {
                        keysValuesToSend.add(kv);
                    }
                }
            }

            if(keysValuesToSend.size() > 0) {
                for(String kv : keysValuesToSend) {
                    String k = kv.split("-")[0];
                    Store.bucket.deleteKeyValue(k);
                }
                ProtocolReceiver.sendMessage(node, Util.getNodePort(node), keysValuesToSend);
                System.out.println("> Key-Value sent to node: " + node);
                return;
            }
        }
    }
    return;
```

Storage/Stabilizer.java

Fault Tolerance:

When a node crashes, the key-value pairs are saved in non volatile memory.

To avoid membership info not up to date, the tasks that change throughout the execution of the service have different delays. Making the priority one the one responsible for the current state of the cluster.

Concurrency

- In many programs, many tasks can be executed in parallel
- Creating a thread for each task is not efficient for a large number of tasks because the large number of threads will oversubscribe the system
- Java provides a mechanism for creating a pool of specific number of threads for executing large number of tasks
- A thread pool keeps all tasks in a waiting queue
- Each thread starts by executing a task from the queue, when done, it grabs another task, and so on

For the context in this project it was used ScheduledThreadPools.

It allows us to have tasks running at specific delay or fixed rate or at the right moment.

It was used to keep the multicast channel and TCP channel open.

To keep the current nodes up to date and to keep the key-values stored in the right Node there are tasks running at fixed delay.

To keep the membership info up to date a broadcast message is sent at a fixed rate of 1 second to all nodes.