Umea University
Department of Computing Science

Advanced Distributed Systems
5DV205

Assignment 3

Date of submission
2018-10-19

Version
1.0

Authors:
Niclas Nyström (c14nnm@cs.umu.se)

Graders:
Cristian Klein ,
Petter Svärd,
P-O Östberg,
Jakub Krzywda
Adam Dahlgren

Contents

1 Introduction 2
2 Usage 2
3 Design & Choices 3
3.1 Communication 3
3.2 Chord & DHT & Keys 3
3.3 Node leaving & Errors 4
4 Discussion 5

1 Introduction

In this assignment the goal was to create a chord-based peer-to-peer system by using the the previous implemented communication protocols as foundation and distributed hash tables. The system should be scalable, resilient and self-managing as desired properties for this assignment.
This assignment uses the REST-solution as foundation for communication and synchronization protocols that was developed in the previous assignment in order to send messages between a server and client.

For this assignment *Java 1.8* was used alongside with the libraries provided in the previous assignment specification. To compile the files *ANT* is required.

2 Usage

There are only one file required to run the solution which acts as both a server and client for each node in the p2p network. That file is called *main.jar*.

In order to compile the files and create the executable *.Jar-file* an ant-script is provided. Furthermore an automation script is provided such that only the command **./build.sh** is needed in order to compile all files into the jar-file (*main.jar*).

The system is executed by typing the following command: **java -jar main.jar my_port known_node_port mode** where **my_port** is the port that this host should use and *known_node_port* is a port of one of the already connected hosts on the network. *Mode* can either be 0 or 1 and 0 is used to construct the network itself i.e. the very first node in a network (in this case the known_node_port should be the same as my_port) and 1 is when a host wants to join a pre-existing network.

Below is an example of how to start the p2p with 4 nodes:

```
./build.sh

// Start the network with a start node on port 10500
java -jar main.jar 10500 10500 0

// Connect into the network above with a node that operates on port
10600.
java -jar main.jar 10600 10500 1

/*
Connect into the network above with a node that operates on port 10700. It
    doesn't matter which node that is known as long as it is connected to the
    network.
*/
java -jar main.jar 10700 10600 1

// The last node.
java -jar main.jar 10800 10500 1
```

When the system is initialized the user is presented with the following options:

- 1. Print all nodes in the network. Used for debugging and is not connected to the DHT implementation.
- 2. Leave the network.
- 3. Print DHT-table.
- 4. Send message to all other nodes in the network by using DHT.

That is executed when typing the number associated with each option.

3 Design & Choices

3.1 Communication

In order for one node to communicate with a other node each node has a *REST*-server and *REST*-client. The concept for the communication are that the server will parse incoming request and convey them to the client and the server (DHT). The client will interact with the user and constantly communicate with other node servers. A illustrative representation can be seen in the figure below:

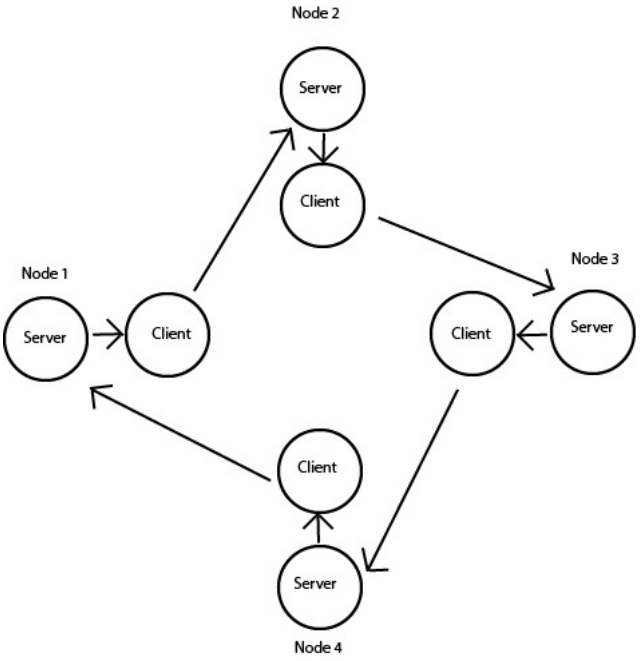

Figure 1: Illustration of Rest communication between nodes and their clients and servers.

Each node server is wrapped inside numerous routes of Rest-protocols that handles messages, nodes, predecessor and successor requests etc.

3.2 Chord & DHT & Keys

When a new node attempts to join a network it will establish a *REST*-connection with a known active node in the network. If the known node responds positively the node can

access the known nodes DHT and compare the keys to find the interval of two keys that the nodes key is between.

If there is a match the node will reroute the predecessor and successor relationships of the nodes that has those keys such that this node is the middle node of those two. In the events of having several intervals that the node are between the node will compare them and find the shortest/narrowest interval. This can happen when many nodes are connected to the network. If there is no match of intervals the node will place it self as the known nodes successor and form a ring of two nodes.

When a connection is established the node will contact all the affected nodes to tell them that the node has connected and synchronize the dht tables and fingers.

The distributed hash table (*DHT*) contains a list of nodes and keys in the network that is close to the node according to $n + 2^i$ where n is the nodes position in the network and i is i:th finger in the dht representing the i:th closest node to n.

The keys are generated by consistent hashing *SHA-1* with a fixed length of m characters (32 as default) using the hosts IP address and port. To compare and analyze the keys there exists methods for comparing common shared digits and calculate distance and intervals between keys.

3.3 Node leaving & Errors

When a node attempts to leave the network it will contact it's predecessor and successor and reroute them such that the predecessors successors is the leaving nodes successor and vice versa. The routing process is similar of the process when a node joins the network, but much more barebone.

In the event of a node leaving unexpectedly the network due to an error the network will trigger it's recovery method. Each node checks it's successor every 20 seconds and if it's unable to communicate with it the node will check it's own DHT and try to find the other node that is connected to the faulty node which has the faulty node as it's predecessor. If it finds it the node will reroute to that node such that it will become the other nodes predecessor and vice versa.

In most cases the node will be successful to regain the connection with the network. But there are cases when it's unable to reconnect and that is when the other node that has the faulty node as it's predecessor isn't stored in the nodes DHT. This happens approximately 1/N of the cases where N is the total number of nodes connected to the network. Lesser numbers of nodes will result in higher chance of it happening. Also, if there are no other nodes connected the network it won't be able to reconnect. In the event of the node being unable to reconnect the program will terminate.

4 Discussion

The biggest problem that I've encountered during this assignment was how to integrate REST as communication protocol. For this type of problem I've usually go with sockets or RMI that was used in the previous course. But I wanted to use REST since that (or SOAP) was the recommendation in the assignment specification. This was the most time consuming since it felt so unnatural to use for this task. Maybe I made things over complicated from the start but I got it eventually to work.

Debugging was also one thing that I found out to be a hassle in this assignment. It was very hard overall to locate bugs and debug them. This problem is one of the characteristic differences between distributed and non-distributed systems so I guess that should be very common problem to face.

The implementation of the system are overall good although there are some things that could be improved such as the error handling part. But I think that the implementation fulfills all the requirements from the assignment specification of being scalable, resilient and self-managing. Each node does only store a fixed amount of nodes thus being scalable and it is capable of handling most errors thus being resilient and self managing.