P2P MidTerm - Report

Student: Andrea Lisi

1 Introduction

The proposed solution is written in *Java*. The project is composed by few classes in order to separate the activities. The major classes are `ChordCoordinator`, which does most of the job, and `Node`, which describes and stores the information of a Chord node.
The coordinator writes a couple of log files in order to have separated information, such as network topology and simulation's history, and a csv file to store all the finger tables of the nodes. This last file is then used by *Cytoscape* in order to visualize the network and perform some static network analysis.

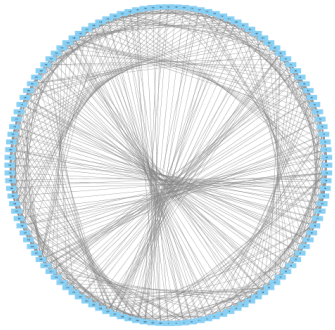

Figure 1: A generated Chord network, 128 nodes, 8 bits

2 Simulation workflow

2.1 The network initialization

First of all the peers have to be created.
To each peer is assigned an identifier generated by applying SHA1 to a random long number. SHA1 generates identifiers 160 bits long and thus each peer would have (with high probability) a really big identifier. To give the possibility to generate smaller networks the hexadecimal SHA1 code is truncated by taking a number of hexadecimal characters equal to number of bits divided by 4. For example, for a 8-bit identifier space only 2 characters would be taken from the generated SHA1, since 2 hexadecimal characters identify $16 \times 16 = 256$ elements, which is equal to 2^8. Once each node has got an identifier the coordinator builds a finger table for each of these nodes. Finally, to each node is assigned its predecessor as Chord's routing expect.

This step produces a log file named *topology**(bits)**b_**(peers)**n.log* where **bits** stands for the number of bits used to create the identifier space and **peers** stands for the number of peers. For example, with a network generated with 16 bits identifier space and 1000 nodes the coordinator creates a log file named *topology16b_1000n.log*.
This step produces also a **csv** file for Cytoscape called ***(bits)**b_**(peers)**n.csv* following the same naming criterion. With the previous setup this file will be called *16b_1000n.csv*.

2.2 Key generations and queries

After the network is built it's time to perform some queries.
For each peer in the network the coordinator generates at random one (or more) key(s) to look for. Zero hops are possible, but bigger is the network smaller would be the probability to perform zero hops.

This produces a log file called *simulation**(queries)**q_**(bits)**b_**(peers)**n.log*, with the usual nota-

tion with **queries** standing for number of queries issued per node. This log keeps information about which node queried which key, who holds the key and the hops followed to reach that key.

3 Analyses and Results

In this section there will be shown some analysis of the simulations. The networks analyzed are built using an identifier space spanned by 12 bits, involving so $2^{12} = 4096$ identifiers.
Three types of network are designed for the experiments: almost empty with 1024 nodes, half-full with 2048 nodes and full.

3.1 Topology analysis

With the help of Cytoscape is possible to show some properties of the Chord network. On the horizontal line of figure 2 there are shown the distribution plots of in-degree, out-degree and clustering coefficient of the nodes. The upper plots are related to the network of 1024 nodes and the below ones to the network of 2048 nodes. The plots related to the full network are omitted because they are reduced to one single bar, since every nodes has exactly 12 incoming-outgoing edges.

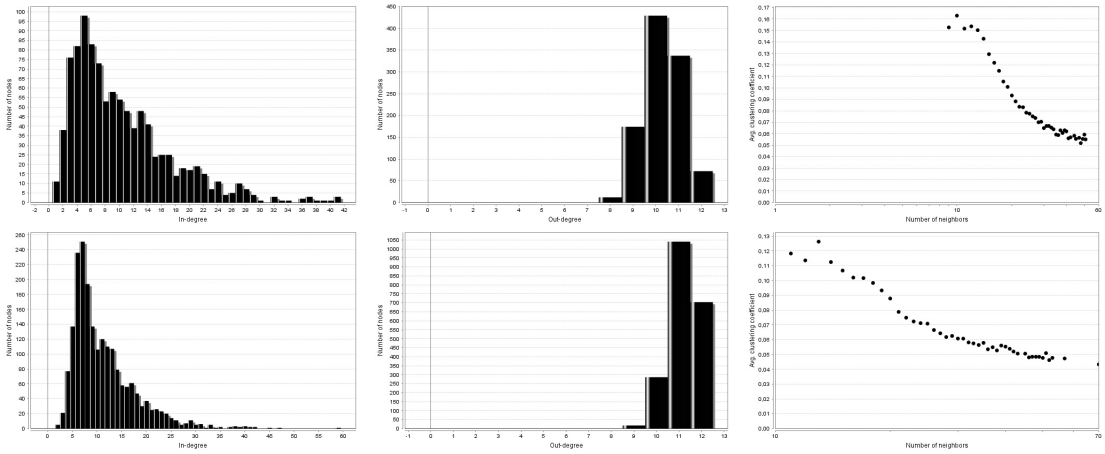

Figure 2: In-Degree, Out-Degree and Clustering Coefficient

The in-degree distribution confirms that in a Chord like network the nodes may be not uniformly queried, since many nodes have few incoming edges, while just a few nodes have much more. Thus, the work distribution may be not balanced between the nodes.
The out-degree distribution tells us that the dimension of the finger tables is actually smaller than $O(\log n)$: in this case by just a few units, but in a larger identifier space (such as 160 bits) this would have a bigger impact on memory saved.
Finally, the clustering coefficient of the network is $O(\log n)$ and the last plot shows how it is distributed.

3.2 Routing analysis

Performing some queries can provide us some data to gather in order to extract some properties of the Chord network. From these data we want to see if we get the expected $O(\log n)$ steps to perform a query.

3.2.1 Distribution of the hops

Figure 3 shows the distribution of the hops needed to solve the queries.
For each chart the x-axis shows the hopCount, i.e. the number of hops needed to solve a query. The y-axis shows the occurrences of the hopCount in the simulation.
Reading the plot table horizontally is possible to see how the distributions change if, for a given chord network, more queries are issued by each node. On the other hand, reading the table vertically gives the idea how the distributions change while increasing the size of the network.

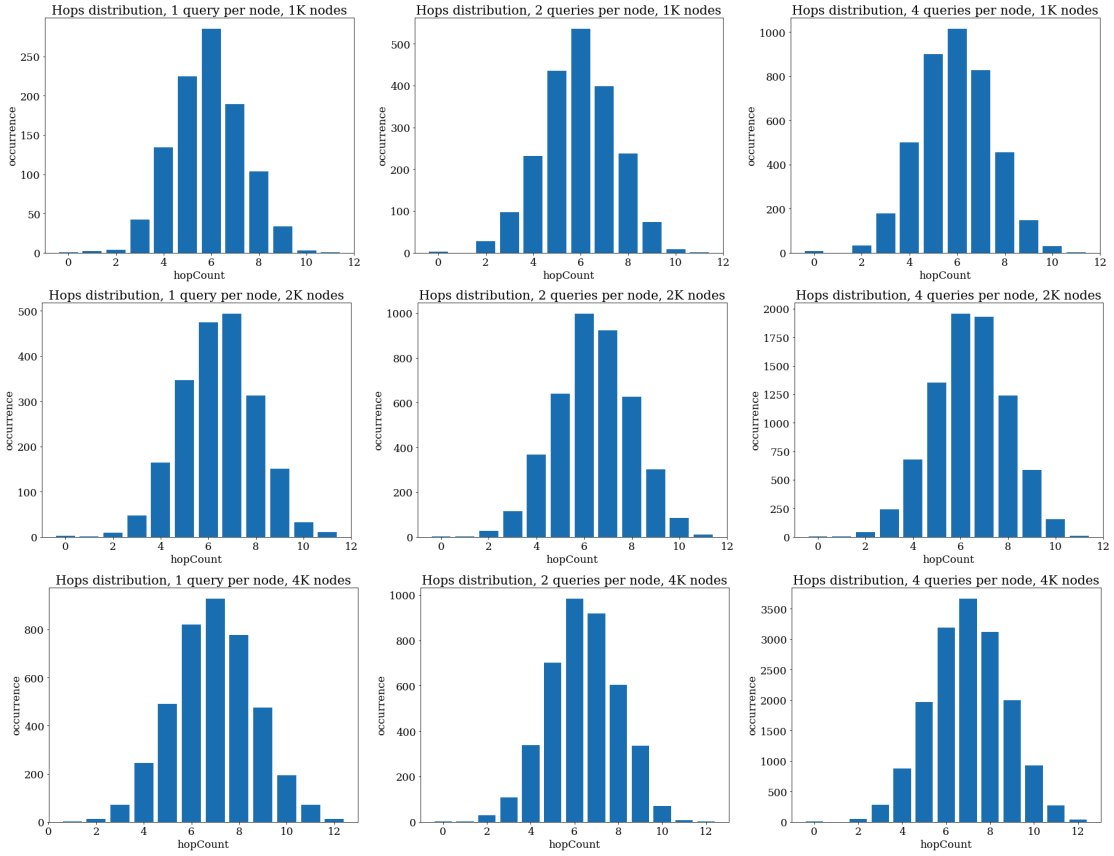

Figure 3: Hops distribution

The peak is reached mostly by the number of hops close to the average (6 or 7 in these cases). Increasing the size of the network increases the hops needed by a logarithmic factor. Increasing the number of queries doesn't affect much the shape of the "bell", and this is good because the number of hops needed (in average) to solve a query doesn't depend on the number of queries.

3.2.2 Distribution of the end-nodes

Figure 4 shows the distribution of the end-nodes, i.e. the nodes which solves the queries.
For each chart the x-axis shows the queryCount, i.e. the number of queries solved by a node during the simulations. The y-axis tells how many nodes solved that amount of queries.
The plot-table follows the same structure of the previous, i.e. increasing the network size on the vertical line and increasing the queries issued by each node on the horizontal line.

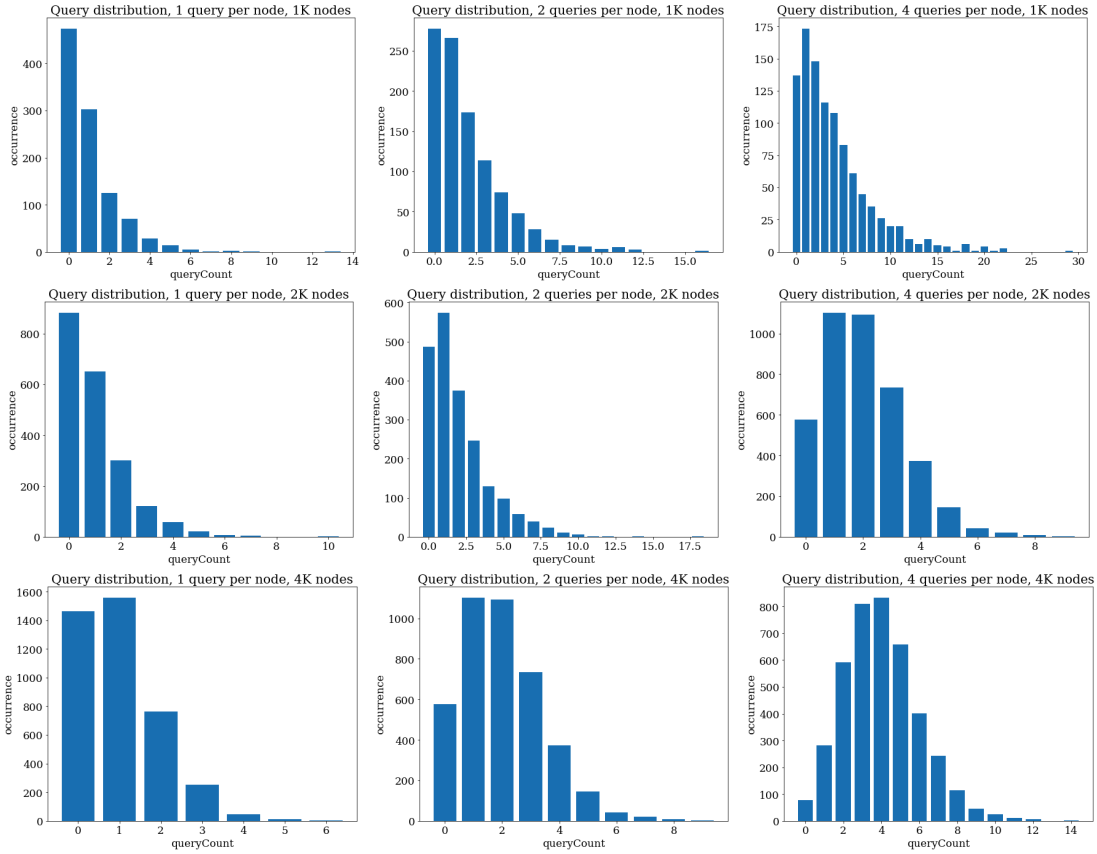

Figure 4: Query distribution

Let's define *maxNodes* as the maximum amount of nodes in the identifier space; *nNodes* as the actual number of nodes in the network; *nQueries* as the number of queries issued by each node.
In average, we expect a node to solve (*nNodes* / *maxNodes*) × *nQueries* queries. In a full network, *maxNodes* is equal to *nNodes* and so we expect a node to solve *nQueries* in average. In a half-full network we expect a node to solve *nQueries*/2 in average, and so on.
It is easy to see that all the charts respect that property: on the lowest line (full network) the peak is reached by 1, almost 2 and 4. On the middle line the peak is reached by 0, 1 (equal to 1/2 × *nQueries*=2) and 2 (equal to 1/2 × *nQueries*=4). Increasing the sparsity of the network these charts tend to assume an hyperbolic shape.
In a real case scenario, i.e. the identifier space is spanned by 160 bits, this distribution may assume an hyperbolic shape also for high number of queries. This confirm us the load unbalancing of a real Chord network, i.e. a few nodes manage too many keys.

3.3 Average number of hops and node distribution

Figure 5 shows how the number of hops (in average) increases as the network size increases. The identifiers space is big 2^{20} and the number of nodes goes from 2^8 to 2^{16}.

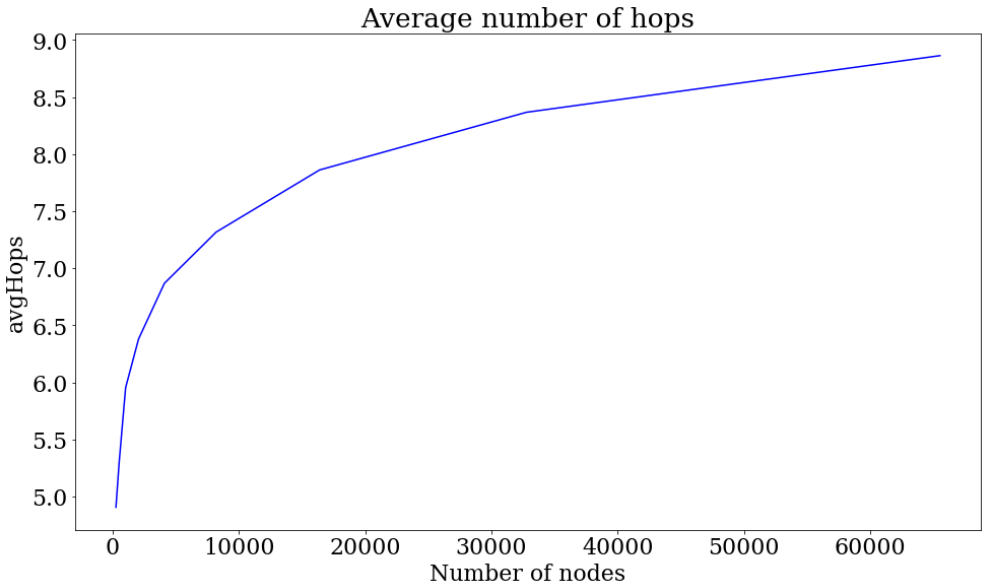

Figure 5: Average number of hops, 20 bits identifier space

This proves that in a Chord-like network the routing steps needed is logarithmic w.r.t. the size of the network. Moreover, this is (almost) equal to $1/2 \times O(\log n)$, as the Chord papers states.

Figure 6 shows how the nodes are distributed, i.e. the gap (in terms of identifiers) between each node and its successor. Both charts show on the x-axis the identifiers and on the y-axis the distance between an identifier and its successor. The above chart represents a network with 1024 nodes, the below one with 2048: the average distance between two nodes is indeed 4 and 2 respectively.

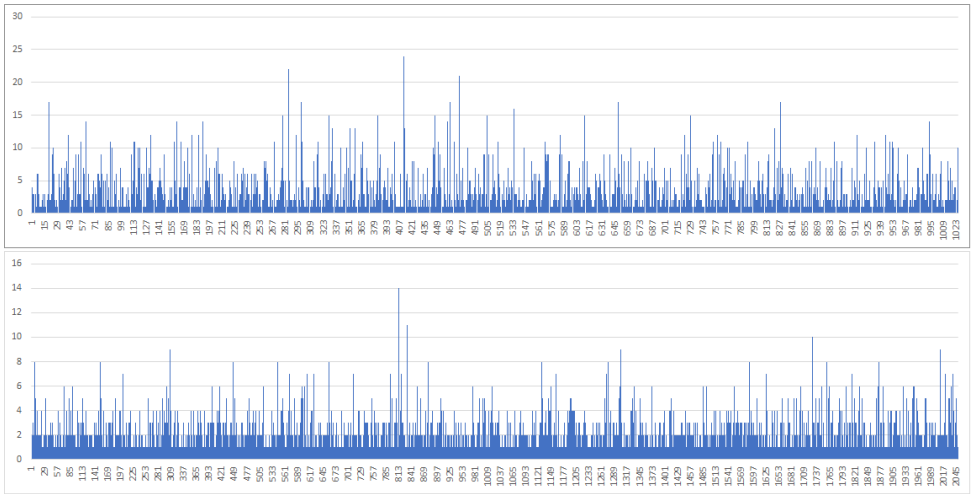

Figure 6: Distribution of the identifiers

User Manual

The classes

The folder *src* contains all the .java files which compose the project:

- **Main.java**: a class with just the main function;
- **ChordCoordinator.java**: the class describing the coordinator. It has two major methods: one to generate the network and one to simulate the queries. The latter one returns some results in a SimulationResults object;
- **Node.java**: class which defines the node of a Chord network. It implements the methods findSuccessor() and closestPrecedingNode(). The implementation of the former is slightly modified since the query isn't distributed but "centralized" on the Coordinator;
- **SHAManager.java**: class which computes the SHA1 of an input;
- **Logger.java**: class with the duty to write a text log file;
- **CSVLogger.java**: class with the duty to write a csv file;
- **SimulationResults.java**: result bundle class;

Note: *the same folder contains also the opencsv.jar file, needed by CSVLogger*

Run the simulation

The ChordSimExe.jar is an already prepared jar file which can be executed with these inputs:

- **b**: bits for the identifier space, $0 < b \leq 160$, multiple of 4;
- **n**: number of nodes in the network, $0 < n \leq 2^b$;
- **queries=1**: (optional) number of queries issued by each node, equal to 1 if omitted, $q > 0$

Example:

```
java -jar ChordSimExe.jar 12 1024

java -jar ChordSimExe.jar 12 2048 4
```

Output

Running the jar produces a few output files in:

- **./log/** it stores the log files, for example *topology12b_1024.log* and *simulation_1q_12b_1024n.log*;
- **./csvLog/** it stores the csv for Cytoscape, for example *12b_1024.csv*;
- **./csvLog/Occ/** it stores the distributions files which produces the plots in figure 3 and 4, for example *b12_n1024_q1_Hops_distribution.csv* and *b12_n1024_q1_Queries_distribution.csv*.