

# Big Data Management

06.05.2018

— **Chord-based simulation of a distributed file system**

Karydis Athanasios

Email: [dsc17008@uop.gr](mailto:dsc17008@uop.gr)

A.M: 17008

Voulgari Eleni

Email: [dsc17005@uop.gr](mailto:dsc17005@uop.gr)

A.M: 17005

MSc in Data Science

2nd Semester

National Centre for Scientific Research 'Demokritos'

University of Peloponnese

## Overview

In this assignment we are going to implement the Chord algorithm in order to simulate the operation of a distributed file system using a dataset from the site thepiratebay. The implementation of the above has been done using Python 2.7 and more specifically the IDE pyCharm.

## Execution Instructions

In this section all the steps needed to execute this project are presented. It consists of the files below:

- main.py which is the executable one
- Chord\_implementation.py which is called by main.py and is the core of the implementation of the algorithm
- func.py which includes all the necessary functions
- Node.py is a class file which is responsible to create all the objects of node, finger tables, etc
- Filenames.txt includes the movies from piratebay.com

In order to be able to execute the project all of the above files must be in the same folder and the user should run the file main.py from the terminal by using the command *python main.py*.

The program asks the user to input the number of requests and then the number of the nodes that will exist in the Chord ring.

With the execution of the project, four .csv files are being created as shown below:

- Average messages - the number of average messages needed to locate a single file
- Load of node by requests - the load of each node in terms of file requests
- Load of node by routed - the load of each node in terms of routing requests
- Sum of messages - the total number of messages that are sent

## Design Choices

The design choices that have been made for the implementation above are:

- The number of requests and the number of the alive nodes are being imported by the user and there is a check if the user inputs integers or not (excluded 0).

- The requests (movies) used to run the Chord are being randomly selected from the file *filename.txt*.
- All the nodes and the titles of the movies (requests) are being hashed by the hashing function `sha1` as the algorithm define. In the beginning we have used to mod the result of the hashing function with  $2^{160}$  but the behavior, in terms of messages, of the algorithm was not right, as the nodes were being increased. Actually with the same number of requests and increasingly number of nodes the graph was fluctuated instead of logarithmic increased as we would expect. We suspect that this happened because the ring was too big and the nodes not so much. So, we decided to mod the result of the hashing function with  $2^{15}$  (32.768) considering the fact that in our experiments the maximum number of nodes and requests used were much less than  $2^{15}$ .
- For each request we produce a popularity measure, using power-law distribution and the algorithm uses this number to search for the specific request as much times as this number defines.
- The function *lookup()*, that searches for a request in the Chord ring, is implemented using recursion. The node from which the search begins is being randomly used and in each run of the *lookup()* the search is being done till the end and the responsible node is being returned.
- In order to make sure that the implementation is right we have conducted some experiments using a smaller ring and less requests. We printed the responsible nodes of the requests and the messages routed and we observed by “running” the algorithm by hand that the results were correct for all the cases we tested.
- For the statistical analysis we created a function that produces all the appropriate measures and writes them in the .csv files mentioned above. We have used these files to produce the graphs and show some relevant results.

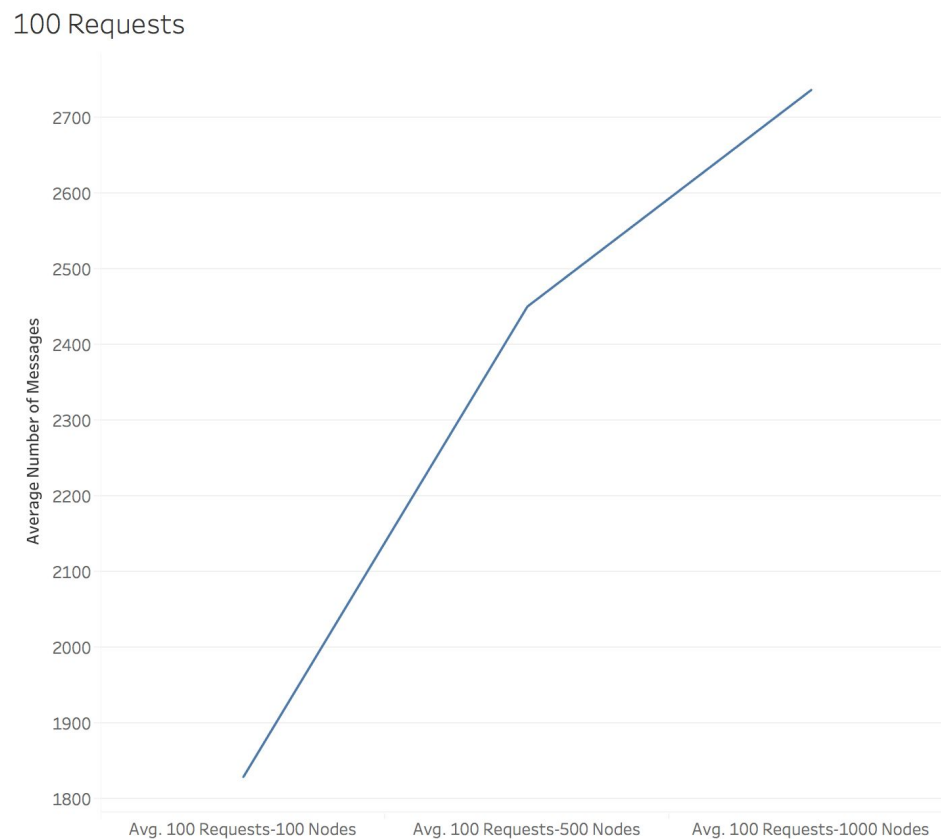
## Experimental Setup

We performed three types of experiments. More specifically, for 100, 1.000 and 10.000 requests, we performed 5 iterations of each experiment with 100, 500 and 1.000 nodes. The total number of the experiments was 45.

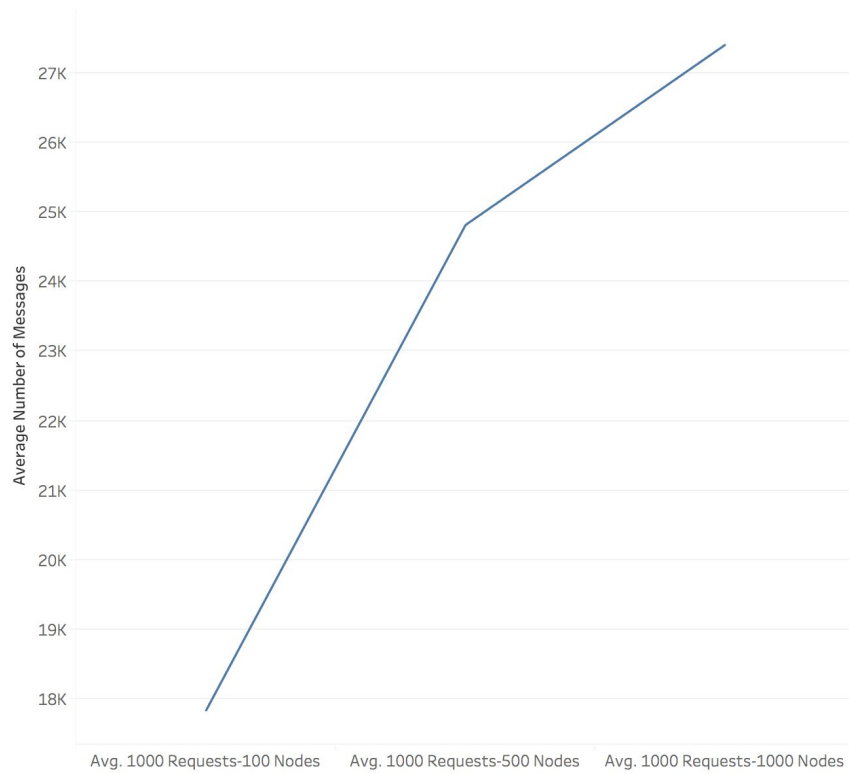
We tried to execute it for 10.000 nodes, but because of hardware limitations, it didn't deliver results in a reasonable amount of time.

## Graphs - Analysis and Results

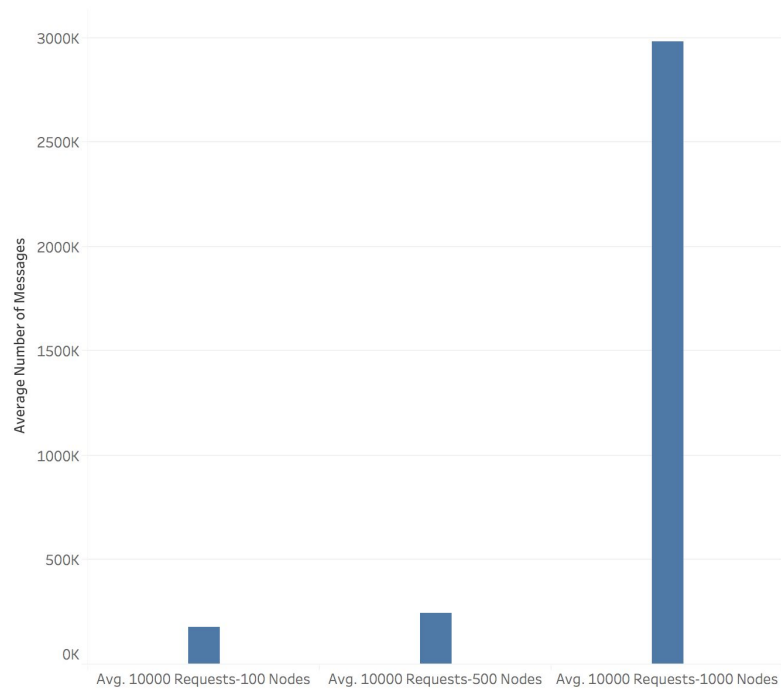
To produce the graphs showing the change in the total number of messages regarding the number of nodes in the ring we used the .csv files which were created by each experiment and created the file *Total\_sum\_messages.csv* which was imported to the Tableau tool.




## 1000 Requests



## 10000 Requests





As we can observe from the produced graphs the number of messages is being increased as the number of nodes increases too. From the first two graphs, we can conclude that there is a logarithmically increase in the number of messages, but as the experiments with larger number of nodes could not deliver results in a reasonable amount of time, there is no certainty about this conclusion. From the third graph the assumption could be that there is exponentially increment in the number of messages, but as the same limitation by hardware applies we cannot be certain without conducting some more experiments.

The load (in terms of file requests) of each node for a given network size can be seen in the *Load of node by requests.csv* file which is produced by the Counter which counts the number of the times a node was responsible for. Because of the randomly selected requests we cannot observe something specific, but it makes sense that the node with the largest number of request responsibility is the node that is responsible for the request that was mostly asked to be served. Thus, the movie with the largest popularity number.

The load (in terms of routing requests) of each node for a given network size can be seen in the *Load of node by routed.csv* file which is produced by the Counter which counts the number of the times a node has routed a message. Again, because of the randomly selected requests we cannot observe something specific, but it makes sense that the node with the largest number of routing messages will be the predecessor node of the node that is responsible for the request that was mostly asked to be served. And this happens because in the majority of the cases, where a specific request is being searched, the algorithm will “visit” the right node’s predecessor in order to find it.

## Possible Solution to the Load Balance Problem

The biggest limitation of Chord algorithm is that there are some nodes that due to excessive workload may crush or delay their response.

In this case we could create virtual nodes which are duplicates of the existing ones, that have the most workload. These duplicate nodes would be “responsible” for the same files that their parent is responsible for and their parent (like a setimel) would assign equally the requests to its children for them to serve. In case the workload is decreased and a child is no longer needed, the parent should be able to erase it.

This way, the overload of specific nodes would be avoided as the requests would be distributed in more serving nodes.

This idea originates from the load balancing in cloud computing.

## Wrap Up - Summary

The above simulation of a Chord-based file search in a distributed file system shows us how files can be spotted more optimal in a peer-to-peer distributed system. The ring and the finger tables allows the search to be carried out via  $O(\log N)$  messages to other nodes.

As for our implementation of Chord algorithm we could say that there are many improvements that can be done. As we mentioned before the lookup searches one request at a time. One improvement, which may reduce the time complexity is to use threads in order to parallelize the workload. In that case every request could be handled by a thread and so the jobs would be executed at the same time. We tried to write clean code but there is always space for improvements. Last but not least we have to mention that we used gitHub because we want to do our work more distributed. You can find the code for our implementation in the link below:

<https://github.com/MscBigDataManagement/ChordImplementation.git>