Chord DHT Relationship

Aldo D'Aquino

1 ALGORITHM

The entire code is written in Kotlin, that is a Java-like statically typed programming language. It compiles in bytecode for the JVM and for this reason is fully interoperable with Java: in a Kotlin class you can use Java objects and vice versa.

I choose Kotlin for it conciseness that drastically reduce the amount of boilerplate code and because with its explicit nullable types and lots of checks helps to avoid errors such as null pointer exceptions.

As IDE I choose JetBrains Intellij IDEA because I consider it more productive than Eclipse and because Kotlin is developed on and for this IDE, but Kotlin is also available as a plugin for the open source Eclipse or as a standalone compiler.

For simplicity I am also attaching the files already compiled for a fast run on the JVM without a Kotlin compiler necessity. Can be executed with "java -cp ChordDHT.jar Main". Use the option "-a" to generate automatically the output topology and statistics files.

1.1 NODE OBJECT

Each node object contains the m-bit identifier, the predecessor and successor node and the finger table, as well as methods for query lookups.

1.2 IDENTIFIERS

A central coordinator generates n nodes assigning them a m-bit identifier.

The identifier is represented as a BigInteger Java class object instead of a bit set to perform operation in an easier way. It is obtained doing the modulo of a SHA1 string to reduce the 160-bit length of the output of the SHA1 to m bit. The input bytes for the sha1 are obtained from a random int generated by the Random Java class.

The use of the module causes two problems: collisions and an uneven distribution. The first problem was solved by detecting collisions and generating new sha1 strings based on always different bytes until more collisions occur. The second problem cannot be solved because it would need a SHA-like algorithm able to guarantee uniformity and to provide m bit outputs at the same time.

1.3 FINGER TABLE

The coordinator also takes care of generating the finger table of each node. This operation is done centrally by looking for the node with the id following the given key instead of simulating the lookup. This allows to obtain the same result with a lower computational cost and to have the system ready for testing in less time.

For a fast search of the following key, the nodes are saved in a TreeMap.

After the finger table is generated, the coordinator plot in a csv file all the edges contained in the finger tables of the nodes, including predecessor and successor node. This is necessary to do some analysis on clustering coefficient for example.

2 NETWORK TOPOLOGY

In the two images below, we can see the topology of Chord network. The first is a small network with 200 nodes, the greatest size entirely displayable, the second one is a partial view of a bit bigger network holding 500 nodes. As we can see the network becomes more uniform as the number of nodes increases.

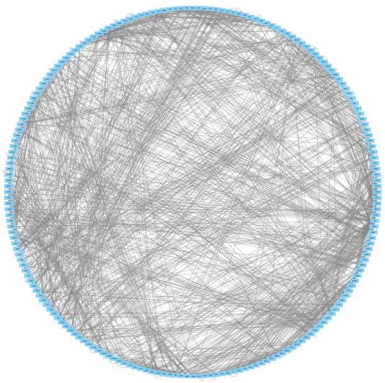

Fig. 1 Representation of a 200 nodes network with 15-bit identifier

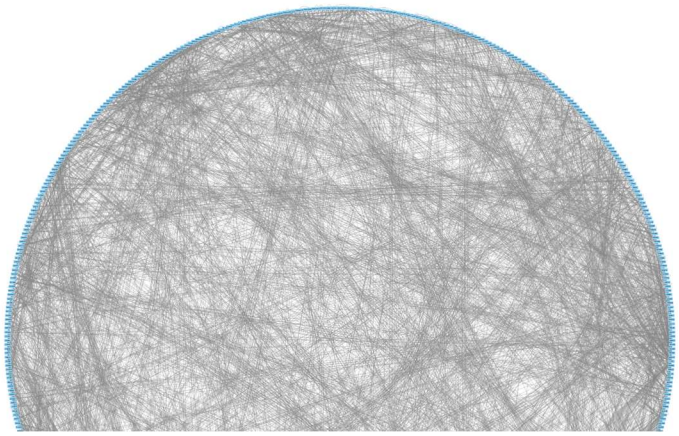

Fig. 2 A partial representation of a 500 nodes network with 30-bit identifier

2.1 DISTRIBUTION ANALYSIS

2.1.1 In-degree

The histogram below shows the in-degree distribution, that is number of ingoing edges, for a network of 32768 (2^{15}) nodes with 64-bit identifiers.

As we expected the distribution can be described by an exponential curve, with a heavy tail. Lots of nodes has a low number of ingoing edges, and few nodes have a higher number of this. The corresponding peers have to manage a high number of messages, and this introduces load balancing problems.

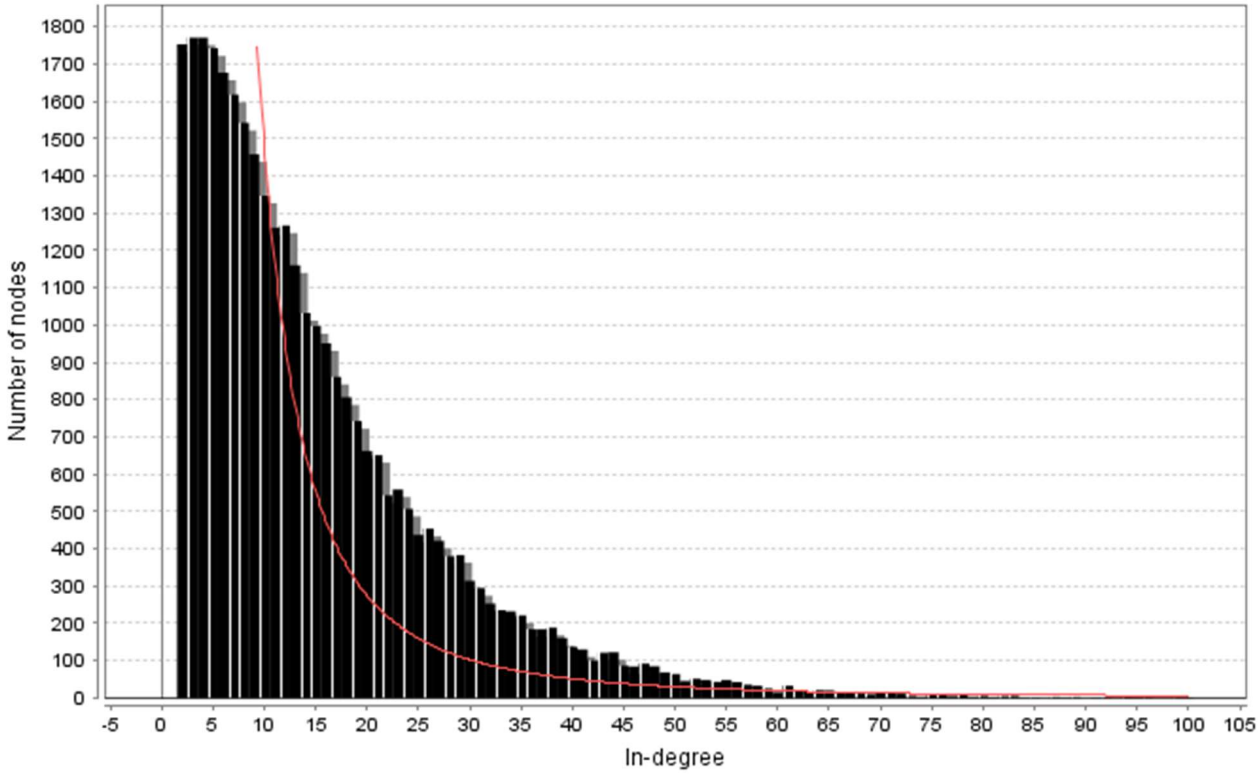

Fig. 3 In-degree distribution, 32768 nodes, 64-bit identifiers.

2.1.2 Out-degree

The histogram below shows the out-degree distribution, that is number of outgoing edges, for a network of 32768 (2^{15}) nodes with 64-bit identifiers.

The istogram has quite symmetric values (with respect to 10.2). In Chord the out-degree is number of entries in the finger table, so each node has approximatively the same out-degree.

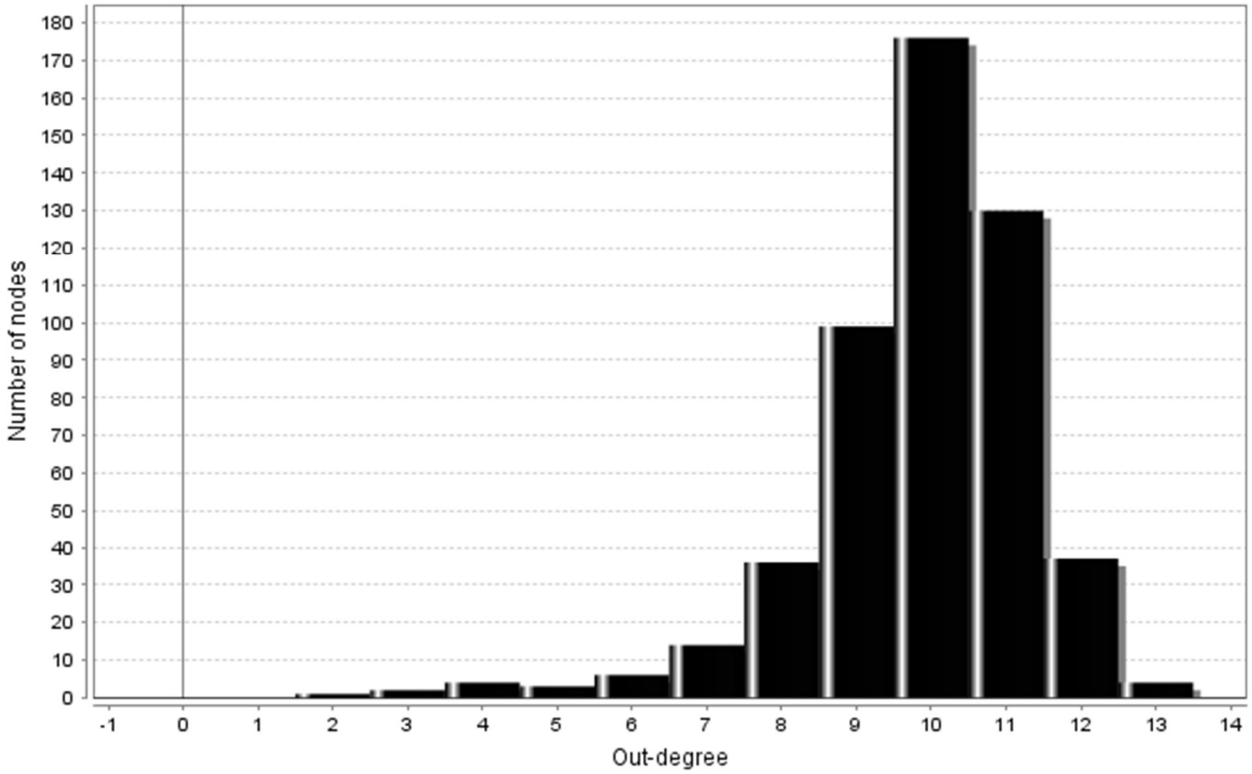

Fig. 4 In-degree distribution, 32768 nodes, 64-bit identifiers.

2.1.3 Local clustering coefficient

The clustering coefficient is the measure of the degree in which the nodes of a chart tend to be connected to each other.

The local clustering coefficient of a node is a measure of how much its neighbors tend to form a clique.

The chart shows the clustering coefficient distribution for a network of 32768 (2^{15}) nodes with 64-bit identifiers.

As for the Global clustering coefficient that decreases as the number of nodes grows, also the local one tends to decrease when the number of neighbors increases.

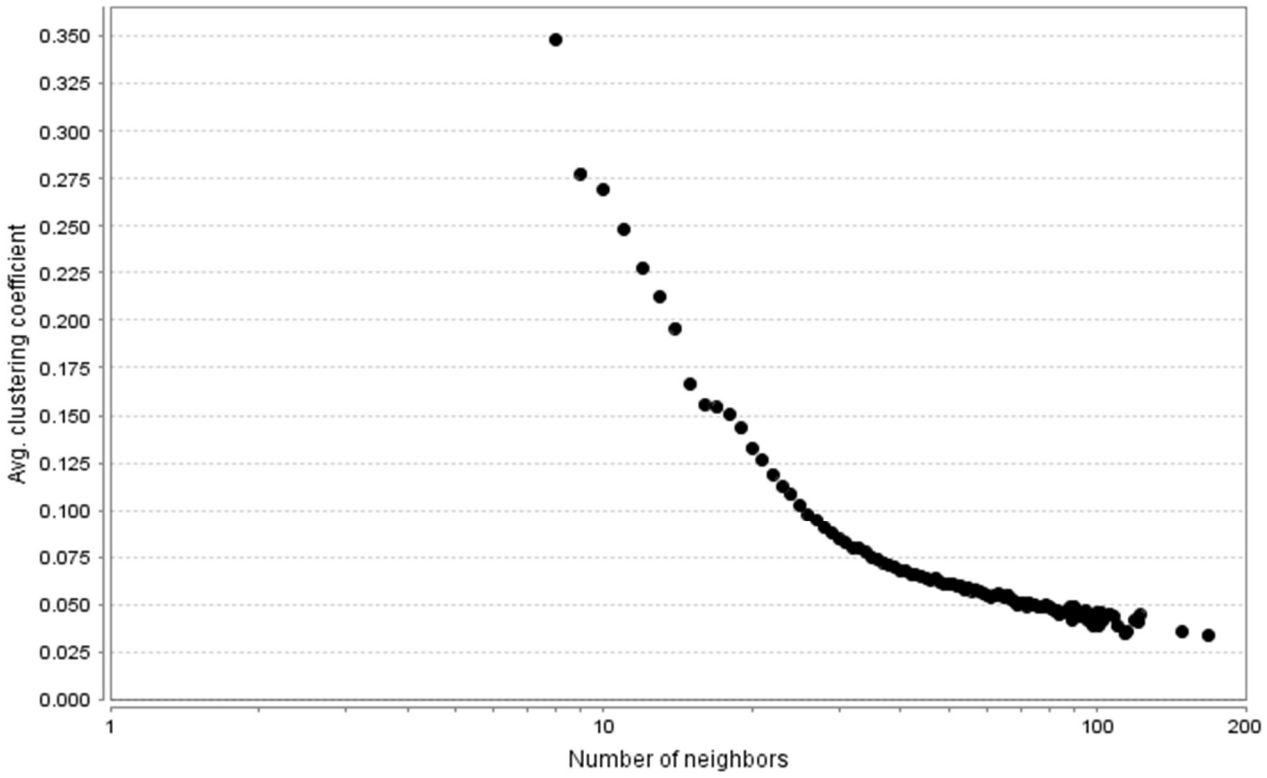

Fig. 5 Local clustering coefficient distribution, 32768 nodes, 64-bit identifiers.

2.2 GLOBAL CLUSTERING COEFFICIENT

The chart below shows the clustering coefficient when varying the number n of nodes in the network. As we can see the clustering coefficient decreases when the number of nodes increases.

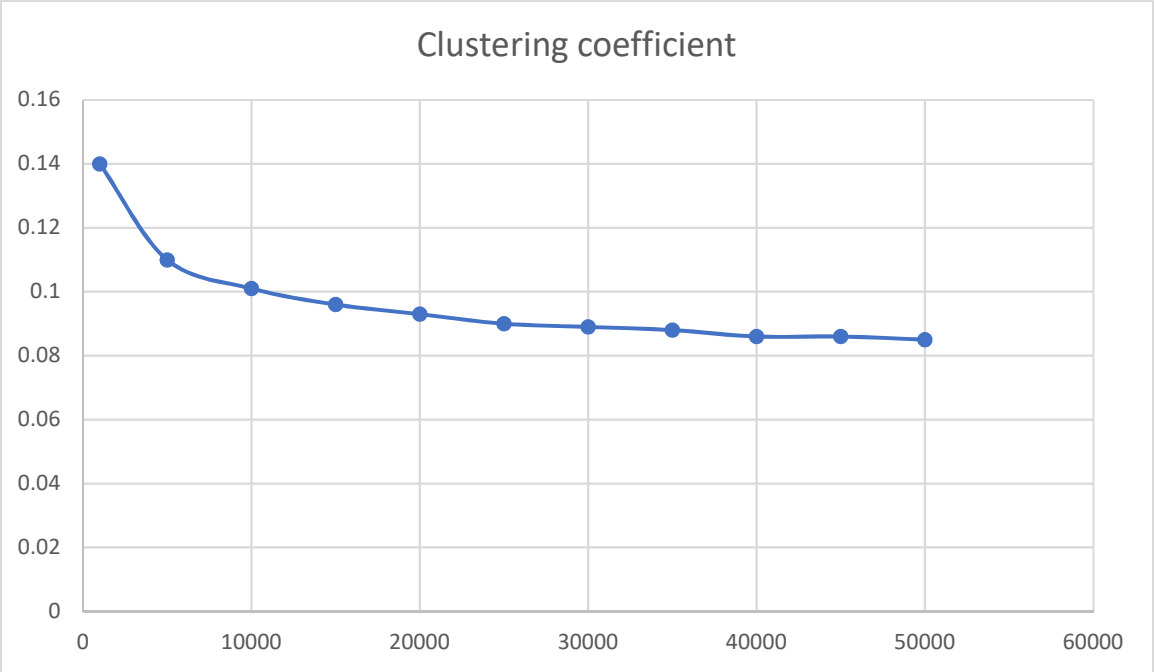

Fig. 6 Global clustering coefficient

3 ROUTING STATISTICS

3.1 NUMBER OF HOPS

This chart shows the average number of routing hops that a query requires to be resolved, that is the average number of crossed node.

We can see that as the number of nodes grows, also grows the hops number. This is because each node has a smaller interval and that produce a power law in the lookup of the pertinence node.

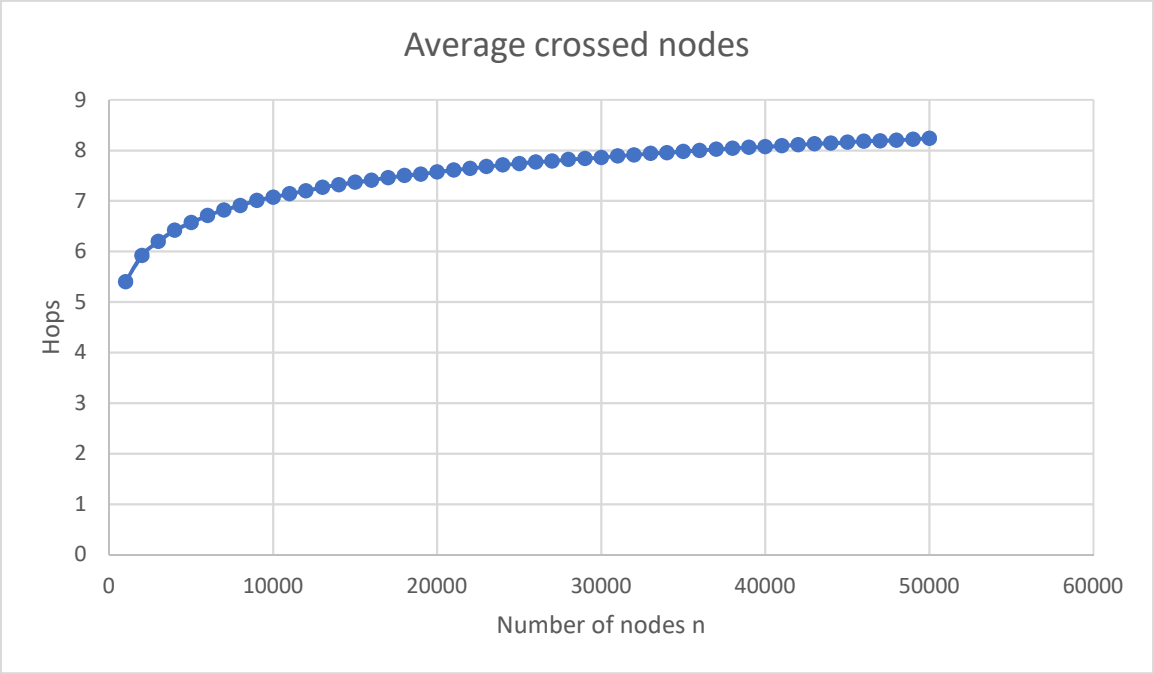

Fig. 7 Number of hops

3.2 QUERIES RECEIVED

This chart shows the average number of queries received by a node: this includes the number of queries resolved, for which the nodes is the destination, but also all the queries that pass throw the node and that the node redirects to another one.

We can see that as the number of nodes grows, the number of queries that pass through a node decrease. The analysis is done with a fixed number of queries, so this behavior is normal because when the number of nodes grows each node has a smaller range of IDs under its control, and so is visited by a smaller number of queries.

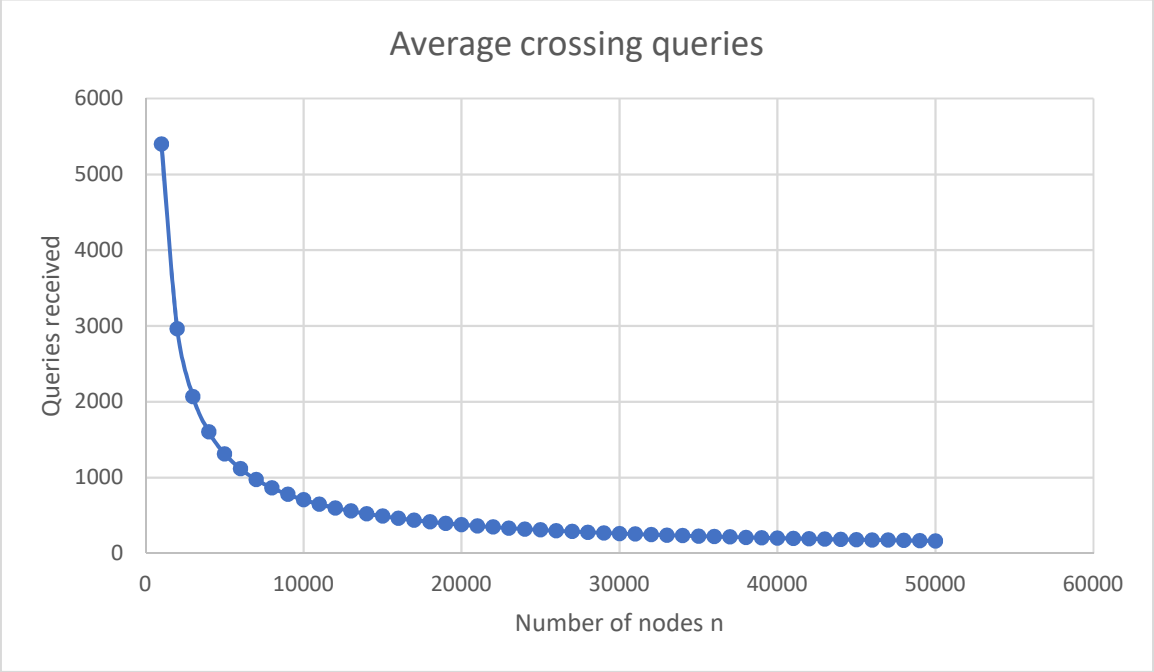

Fig. 8 Queries received

3.2.1 Distribution

The chart below shows the distribution of queries. On the X-axis we have the ID of the node and on the Y-axis the number of queries that each node receives.

Except for a little 'noise, we have a fairly homogeneous distribution. This is important because means that the load is well balanced.

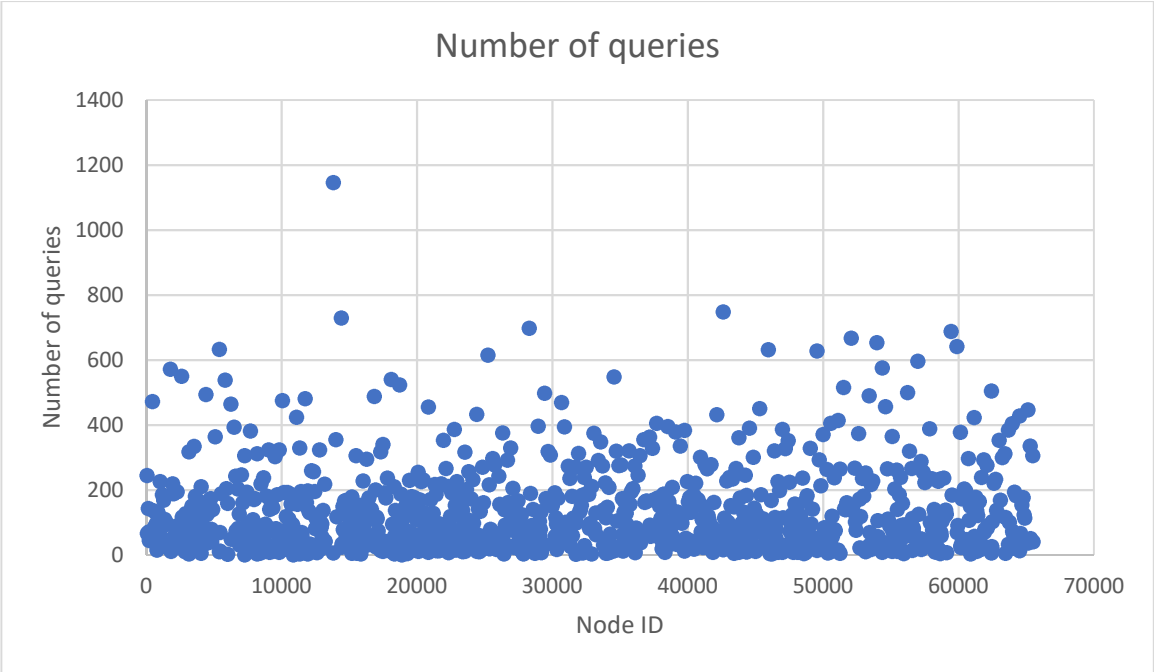

Fig. 9 Queries distribution