# Analysis of the Billion Triples Dataset using Hadoop

Cristian Di Pietrantonio, 1604142

June 29, 2017

**Abstract**

In this work the Hadoop framework is used to answer several questions about a huge graph representing a portion of the semantic web. Several experiments are performed to assess the efficiency of the implementation proposed. The Hadoop program is run on a local cluster using 30 million triples (the first 3 files of the dataset).

## 1 Introduction

The *Billion Triples Dataset* represents a part of the semantic web. It is a collection of triples of the form $(s, r, o)$, where $s$ and $o$ are two resources and $r$ is a relation standing between them. The dataset can abstracted as a graph where the nodes represent resources, and there is an edge between two nodes for each existing relation between the corresponding resources. Additionally, each edge may have one or more annotations, called context, that tell where that information (i.e. the existence of the relation) comes from. Finally, a node is said to be "blank" if it does not have a URI.

In this work we perform several analyses on the dataset. In particular, we are interested in answering the following tasks and questions:

1. Compute the number of distinct nodes and edges in the corresponding RDF graph.
2. Compute the outdegree distribution: does it follow a power law? Plot the result in a figure.
3. Compute the indegree distribution: does it follow a power law? Plot the result in a figure.
4. Which are the 10 nodes with maximum outdegree, and what are their respective degrees?
5. Compute the percentage of triples with empty context, the percentage of triples whose subject is a blank node, and the percentage of triples whose object is a blank node.
6. Each triple can appear with different contexts in the dataset. For each triple, compute the number of distinct contexts in which the triple appears (the empty context counts as 1). Report the 10 triples with the largest number of distinct contexts (break ties arbitrarily).
7. Remove duplicate triples. How much does the dataset shrink?

The high number of triples makes impossible to perform calculations with traditional algorithms. Instead, we are going to use the Hadoop framework that allows us to write programs that perform a distributed computation.

## 2 The algorithmic idea

A brief overview of the algorithmic idea is now presented, so to introduce the reader to the intuitions behind the detailed algorithm shown in the next section. During the development, it has been important to keep in mind that the seven tasks share similarities and are not totally disjoint. Answering to one question can lead us closer to the answer of another question. As a start, consider point 1. We can answer to the question using a closed formula once we know the indegree (or outdegree) distribution of the nodes (point 2 and 3); in order to know the number of distinct nodes, simply sum up the frequencies of the various degrees (since 1 unit in frequency means one node having that degree). A similar approach can be taken to know the number of distinct edges. Of course, we should compute the distribution correctly, that is, without counting the same node twice.

Observe now that answering to questions 1 to 4 can be faster and easier once we remove duplicate triples. Also, points 5 and 6 can be answered *during* the process of duplicate removal; this is true because we can *map* a context and triple properties (e.g. is the triple's subject a blank node?) to the corresponding triple, so that a triple becomes a *key*, and as such is unique (duplicate removal). Doing so, we have the necessary information at the next step to answer questions 5 and 6.

It seems reasonable, then, thinking of a workflow that answers to question 7 first, then 5 and 6; next, 2, 3 and 4. Finally to question 1.

Before we proceed to the algorithm description, some clarifications on data and questions interpretation must be done. These are "personal" interpretation which try to meet at best the intended meaning. In question 5, we are asked to compute the percentage of triples that satisfy some property. In this work we assume that in the total count of the triples, each triple contributes in the number of its duplicates (in short, we count the duplicates too). This is because we are asked about *triples* and not *nodes* in the graph (i.e. how many blank nodes are in the graph); each triple is a line in a file and it is independent from the other.

## 3 The Hadoop program

The designed hadoop program can be divided in 4 logical parts, realized in practice with 6 jobs. What follows is the algorithm description. In each point we describe a job and its goal, using the natural language so that it is easier for the reader to follow. In most cases the code behind can be easily deduced. When not, a more detailed explanation is presented.

1. The first job associate to each triple the number of reported contexts (i.e. the number of duplicates), the number of *distinct contexts*, and the number of empty contexts associated with it (it isn't specified if there can only be one or more, so we expect even more than one). This answers to question 7 (since the triple is the key, hence it is printed only once). The output looks like

   Subject, Relation, Object, num_duplicates, num_distinct_contexts, num_empty_contexts

   (commas added for clarity).

2. The second job takes as input the output of the first job and associates to each count of distinct contexts the triples that have that value as count of distinct contexts. Obviously, in this case the keys are nonnegative integers and hence are sorted (answer to question 6). Since, for each triple, the input file contains the number of associated empty contexts, we

can map to a special key, $-1$, the number of empty context encountered. Also, since we have access to the number of duplicates for each triple and we can tell if there is blank subject or object, we map to key $-3$ and $-2$ the number of triples having blank subjects and objects respectively. Each triple can have at most one blank subject and one blank object; but a triple has $d$ many duplicates (total number of contexts), so we map $d$ to $-2$ if a triple has blank object, and $d$ to $-3$ in case of blank subject, for some $d$. During the reduce phase these values are summed up and give the total number of triples that have a blank subject and the total number of triples that have a blank object (question 5). The output file looks like this:

-3 num_triples_blank_subj
-2 num_triples_blank_obj
-1 num_empty_contexts
23 node_x
23 node_y
45 node_z
...

Notice that, while the key is always an integer, the first three values are integers, but must be written as strings because the other values are strings representing node identifiers.

3. The third, fourth and fifth jobs are used to compute the indegree and outdegree distribution (questions 2 and 3). The third job takes as input the shrunk dataset computed by job 1 and, for each triple, it assigns to the object node the label "i" (incoming edge) and the label "o" (outcoming edge) to the subject node. In the reduce phase, then, for each node there is a list of values composed by "i"'s and "o"'s. From this list we compute the indegree and outdegree of each node. What has to be done now to obtain a distribution is to have a list of possible indegree and outdegree values, and to associate to each of them the number of nodes having that degree. The output file contains lines like the following one:

node_x, indegree_value, outdegree_value

4. Job four takes as input the output of the third job (a list of nodes with associated their indegree and outdegree) and associates to each indegree and outdegree values encountered the nodes that have that indegree and outdegree. In this way we have the distributions asked by question 2 and 3. It must be noticed that we can't just use integers as keys because we have to distinguish between indegree and outdegree values. So keys are Text values, having has fist character "i" or "o" according to the type of degree and then the value follows. The reducer finally compute the frequency for each key (i.e. degree). The output of this job looks like this for, say, outdegree 42, that has frequency 20, and indegree 34, that has frequency 300:

i34 10
o42 300

We could have called it done here for what concerns the computation the degree distributions. However, certain node degree values can be more common than others, and some reducers that have to process those common degree values can have a long list of associated nodes to go through to compute the frequency. To have a better balancing of the computational work, we can think of a way to split the values associated to a key among

a set of "sub-keys" obtained from the original key. If $[i|o]$[DEGREE_VALUE] is the pattern used for the original key, we add yet another component by appending the hash value of the node we are currently considering (remember, the input to this job is "NODE INDEGREE OUTDEGREE"), for some simple hash function; so the actual key used in this job is $[i\|o]$[DEGREE_VALUE][H(NODE)]. The output of this job would be like the following.

```
i34_389 5
o42_394 150
i34_865 5
o42_124 150
```

See that, in contrast with the previous example, the outdegree values 42 has its frequency partitioned in two subkeys which differ by the hash part following the underscore.

5. In Job 5 we compute the final distributions using the output of the fourth job (the partial frequencies). The map function reads a line, gets the original key by removing the suffix beginning with the underscore, and associates to it the corresponding values. These values are then summed in the reduce phase to give the complete frequency for each degree value (answers to 2 and 3). We can now use one of the two distributions, let's say the indegree distribution, to compute the number of distinct nodes $D_n$ and the number $D_e$ of distinct edges. All without using Hadoop.

$$D_n = \sum_{v \in V} f(v)$$

where $V$ is the set of possible values for the indegree of a node and $f(v)$ is the frequency of value $v$ in the graph (i.e. the number of nodes with indegree equal to $v$). Next,

$$D_e = \sum_{v \in V} v f(v)$$

That is, the number of edges is the sum over all possible indegree values of the product between a indegree valuea and the total number of nodes with that indegree.

6. Lastly, in Job 6, we answer to question 4 by using the output of job 3. Now only the outdegree of a node is considered; each node is associated to its outdegree. Concretely, the map function writes the outdegree as an integer key and the node as text value. The runtime sorts pairs by key and when the reducer writes each (outdegree, node) pair on file, these are ordered. To know which nodes have the top 10 greatest outdegree, it is only needed to look at the last 10 lines of the output file.

The algorithm just illustrated is implemented in Hadoop. A single reducer is used in all jobs, since we are using a single machine. Reducers in jobs 4 and 5 are are used also as combiners, since they sum up values, so that to reduce the quantity of data shuffled. The answers to the various points are then listed in the next section.

# 4 Output of the program

The first job performed removes and keeps track of duplicate triples in the input file. As a result, the output file is substantially smaller than the input one. As matter of fact, the first three input files together occupy 6.3Gb of secondary memory; after the duplicates are removed, the space

occupied drops to 1.6Gb, a 75% reduction. Furthermore, the shrinking improves future jobs' performance, since there is a much less quantity of data to move between processes and nodes, especially in the shuffle phase.

The second job lets us know that the ten triples with the largest number of distinct context is 1690. To see the top 10 triples with the largest number of distinct contexts, look at the file "top_10_distinct_contexts.txt" (they are in a separate file for fomatting reasons). In the same output file, the first three rows contain the special keys $-3$ associated to the number of triples with blank subject, $-2$ for the number of triples with blank object and $-1$ for the number of empty contexts.

| | |
|---|---|
| $-3$ | 1927756 |
| $-2$ | 409647 |
| $-1$ | 0 |

We can immediately see that the number of triples having blank context is zero. Almost two million triples have a blank subject, while the count decrease substantially for the triples having a blank object.

The third, fourth and fifth jobs compute the indegree and outdegree distributions. What comes out is that they have respectively 1009 and 438 distinct degree values (each of them having an associated frequency). Figure 1 and 2 show the plot of the distributions in log scale. On the y-axis we have (the logarithm of) frequencies; points on the x-axis represent degree value indexes (again, we take the logarithm of these values). Points on the x-axis are in decreasing order of frequency. Both plots shows clearly a linear relation, allowing us to say that degree values and frequencies are bound together through a powerlaw.

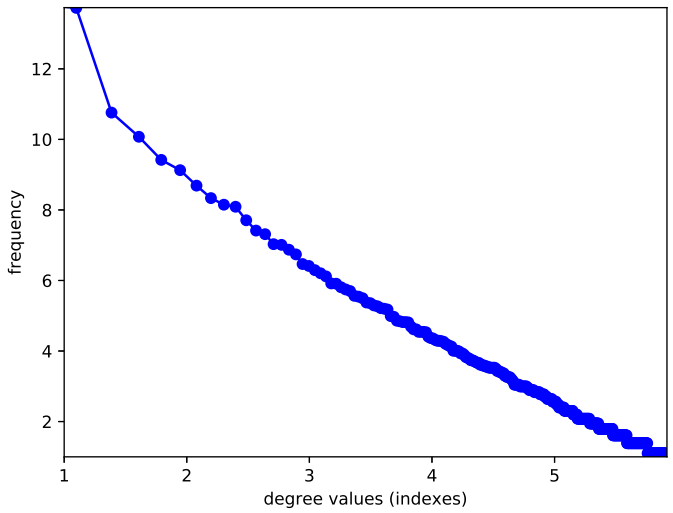


Figure 1: Plot of the indegree distribution in log scale.

Once the two distribution are computed, any one of them can be used to find the number of distinct nodes and edges using the approach illustrated in point 5 of the algorithm (previous

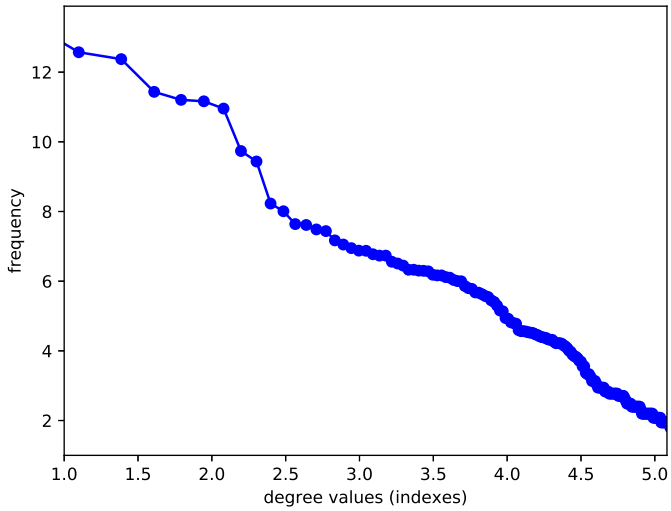


Figure 2: Plot of the outdegree distribution in log scale.

section). In the analyzed part of the graph are present 4622167 (4 million and 6 hundred thousand) nodes and 10942593 (ten million nine hundred thousand) edges.

Finally, the sixth job compute the 10 nodes with the largest outdegree, that are reported in the file "top_10_outdegree_nodes.txt".

# 5 Performance analysis

Now we turn our attention to the efficiency of the algorithm proposed. Hadoop is run on a single machine with 1TB disk, 12GB ram, i7 quad core (8 threads) processor with maximum frequency 3.4Ghz. Jobs are run sequentially, thus can be useful to look at their running time and memory footprint to determine which are the most expensive. Figures 3 and 4 show the six jobs' running times (in seconds) and memory usage. Table 1 shows the exact values. The total time of execution is of 1168 seconds $\approx$ 20 minutes.

| Job | Time (s) | RAM Usage (Gb) |
|---|---|---|
| 1 | 830 | 10,11 |
| 2 | 154 | 2,6 |
| 3 | 97 | 2,67 |
| 4 | 32 | 0,52 |
| 5 | 20 | 0,31 |
| 6 | 35 | 0,52 |

Table 1: Running time and memory usage of each job.

As one could have imagined, the first job that deals with duplicate takes substantially more time, 830 seconds, and memory that the others. Once the duplicates are removed, the processing time diminishes right away, already at the second job, dropping at 154 seconds. Also, 8 gb of

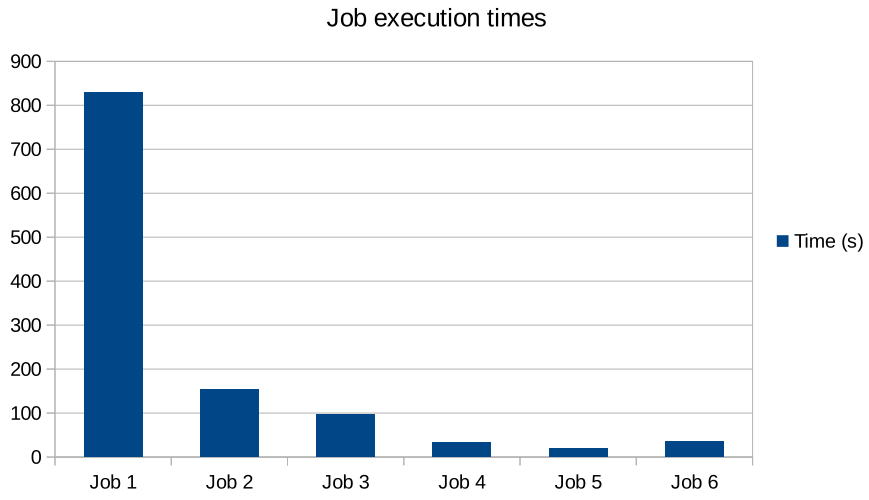


Figure 3: Comparison of jobs' running times.

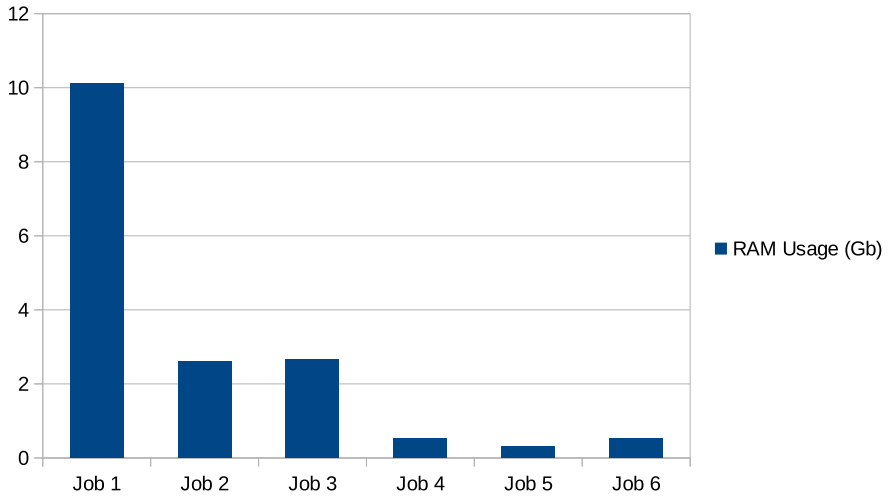


Figure 4: Comparison of jobs' memory usage.

memory gets released. All other jobs complete in one or two minutes, showing that the approach used, and most of all the order in which questions are answered, is a valid one. Shuffle phase is expensive, so in each job we try both to

- maximize the useful information that can be extracted from data (e.g, in the first job we associate to each triple three numbers), and
- to minimize the quantity of data necessary to represent this information (e.g. in the first job we throw away all contexts, keeping just three integers for each triple).

In this way, only the minimal amount of data is exchanged among nodes in the cluster. As

last note, the decision to split the degree distribution computation in several jobs should give a payoff for when more input files are used.

# 6 How to run the code

All the program is contained in a single file, Homework.java. It must be simply compiled and run. It does not require input parameters, since the input path is hardcoded (for a pratical reason, to avoid every time to write the parameter). All the input files must be put in the hdfs folder */input*. The program then takes care of creating the output folder corresponding to a job (e.g. *output6* for job 6) and use them as input to subsequent jobs.