Fabio Luchetti – 492470

Distributed Enabling Platforms

The PageRank problem: a Hadoop implementation

Project Report

Table of Contents

Project Specification

The aim of this project is the design, implementation and evaluation of the PageRank algorithm, a well-known mathematical procedure at the core of many big-data problems dealing with information network analysis.

The implementation has been done using the Hadoop/Java framework, thus leveraging MapReduce programming model. A private cluster on top of AWS EC2 free tier instances has been built and configured on purpose, in order to run tests and experiments in a real distributed scenario.

General problem definition

A complete treatment of the PageRank algorithm is beyond the scope of this report, so the interest reader is referred to the rich literature available on the web. For us, in order to tackle the problem, a simple - but not trivial - definition will be sufficient. Let's consider a set of N webpages $\{p_0, p_1, \dots, p_{n-i}\}$ and their linking structure $\{l_0, l_1, l_2, \dots\}$ (the hyperlinks). This can be viewed/modeled as a directed graph with vertices P: $\{p_0, p_1, \dots, p_{n-i}\}$ and edges L: $\{l_0, l_1, l_2, \dots\}$. We enumerate the webpages, and associate with page p_j a number between 0 and 1, called the PageRank of p_j: the vector r: $\{r_{p0}, r_{p1}, \dots, r_{p\_n-1}\}$ of PageRanks is a probability distribution, and in a broad sense measures the importance of the pages in the collection; the bigger the PageRank, the more important the page. The values r_{pj} are defined as:

$$r_{p_i} = d \sum_{p_j \in B_{p_j}} \frac{r_{p_j}}{|p_j|} + (1 - d)$$

where

- d is the so-called damping factor
- B_{pj} is the set of backward links of p_i (i.e. links to p_i)
- $| p_j |$ is the number of forward links of p_j (i.e. links from p_j)

The mathematical foundation of this formula bank on the theory of stochastic processes, by interpreting as an ergodic Markov chain the likelihood that a random surfer, navigating the web through hyperlink, will arrive at any particular page. But more in general the algorithm may be applied to any collection of entities with reciprocal references.

Proposed algorithm: design choices and parallelization

To calculate the ranks vector, we adopt an iterative approach, in the limit of the number of iteration. A short comment is due before proceeding: please, note that Hadoop is definitely not suitable for iterative processing. The design goal of its creator were different and meant primarily for resilient batch processing of data with (possibly) a single linear scan of the input files, on commodity machines. Instead, here we require that the results of each iteration are written to disk and then read again (even on the same node) over and over the next iterations. By contrast, other technologies such as Spark or similar do exists and are best suited for faster Map Reduce iterative processing, ***provided you have*** sufficient memory and are not working on too large dataset. In the following, the trivial details of the implementation are voluntarily avoided and, if ever necessary, a proper explanation of the business logic of the application is contained directly in the source code.

The work is split in four different MapReduce jobs, where each output is the input for the subsequent job: ParseGraph, CalculateRank, CheckConvergence, SortRank. With more detail:

ParseGraph, Job 1

The ParseGraph job, merely captures the outgoing links from each of the N page of the collection. Each outlink is represented in the input file by a line with two integer value. As output, it stores for each page encountered: the page itself, its initial ranks and its outgoing links. We start from an initial value of 1.0 for every page (to be coherent with the definition of probability distribution, it should have been 1/N, but this unnecessarily complicate the algorithm without affecting the final result).

CalculateRank, Job 2

The CalculateRank job constitute the most expensive part of the program, and it should calculate the pageranks using a simple iterative algorithm. The job keeps being iterated (upon updated values) unless a specific convergence criteria is met, or the maximum number of iteration has been reached. For a page p, its PageRank at iteration k+1 is calculated as

$$r_{k+1}(p_i) = d \sum_{p_j \in B_{p_j}} \frac{r_k(p_j)}{|p_j|} + (1-d)$$

$$r_0 = 1.0$$

Let's break it down into sections:

- r_k: each page has a notion of its own self-importance.
- $|p_j|$:each page spreads its vote out evenly amongst all of its outgoing links. |pj| represents the count of outgoing links for page pj.
- $r_k/|p_j|$: so, if a page p has a backlink from page j, the share of the vote it will get is $r_k/|p_j|$
- d: is a double constant (by default set to 0.85), called damping factor. All these fractions of votes are added together but the total vote is "damped down" by multiplying it by d.
- (1-d): if divided by N, this term should make the sum of all the ranks of the pages - meant as a probability distribution - sum to one. It also accounts for the pages that have no links to it (no backlinks), so that they still get a small rank of (1-d).

CheckConvergence, Job 3

As long as the computation keeps going, it could happen that before the last iteration no significant changes applies to the ranks vector. So, to shorten the whole completion time, every some iteration a convergence test is done upon the current value of the ranks: if the distance in norm L_1 is below a certain accuracy value ($\sum|r_{k+1}(p_j) - r_k(p_j)| < \varepsilon$), the PageRank algorithm stops iterating.

SortRank, Job 4

The SortRank job lack the Reduce phase, and take advantage of the Shuffle and Sort subroutine provided by Hadoop to simply reorder and associate each page with its final PageRank value.

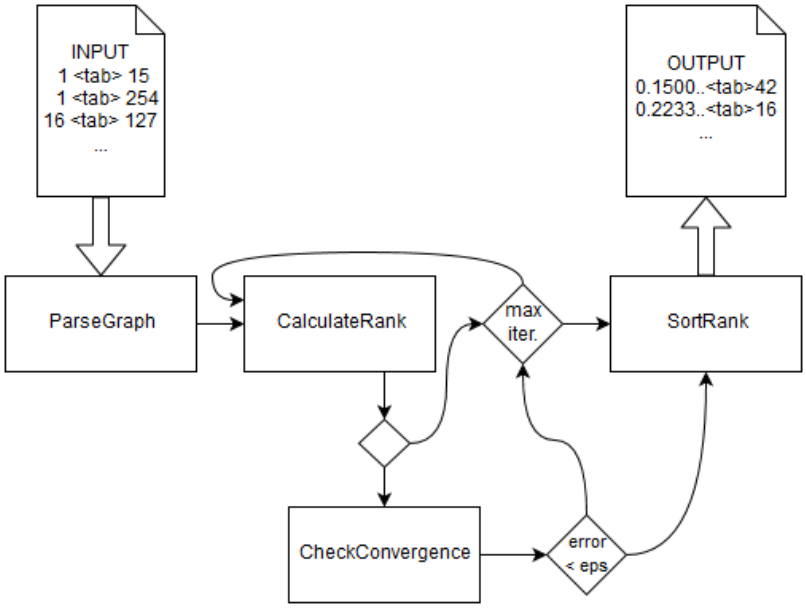

Fig. 1: Workflow chart of the application

User manual

You will find in the zipped folder all the necessary files to compile and run the developed solution, provided you have settled up a working Hadoop environment. To compile the sources, we exploit Apache Maven building facilities.

Compile

In order to compile, cd' in the project directory and simply run:

```
mvn clean package
```

Run

The program make use of a set of configurable variables, which have to be passed in input at application launch through the proper flag. These are:

--input (-i)	<input>	The directory of the input file (or the file itself). [REQUIRED]
--output (-o)	<output>	The directory of the output result. [REQUIRED]
--damping (-d)	<damping>	The damping factor [OPTIONAL]. Default is 0.85.
--count (-c)	<max iterations>	The maximum amount of iterations [OPTIONAL]. Default is 3.
--accuracy (-a)	<accuracy>	The estimate of error in norm 1 for the rank vector [OPTIONAL]. Default is 1.0.
--periodicity (-p)	<periodicity>	Checks for convergence every <p> rank-calculation iterations [OPTIONAL]. Default is 2.
--help (-h)		Display an help text.

Finally, to run the application:

```
hadoop jar target/pagerank-1.0-SNAPSHOT.jar pad.luchetti.pagerank.PageRank -i <input>
-o <output> […]
```

You can pick a dataset with the proper input format from the publicly available Stanford web graphs collection.