Università di Pisa

Artificial Intelligence and Data Engineering

Cloud Computing

PageRank

Project Documentation

TEAM MEMBERS:
Federica Baldi
Daniele Cioffo
Edoardo Fazzari
Mirco Ramo

Academic Year: 2020/2021

Contents

1 — PageRank

1.1 Introduction

In this section a description of the MapReduce implementation of *Page Rank* is given. The algorithm is carried out in **four distinct steps**:

1. *Nodes counting* phase
2. *Graph Construction* phase
3. *Page Rank Computation* phase
4. *Sorting* phase

We have decided to represent the structure of the graph through adjacency lists, so each node (that represents a page) will keep the list of outgoing edges as status information, as well as its ranking.

1.2 First Phase: Nodes Counting

To compute PageRank the total number of nodes is required. Considering that the number of nodes is unknown at the beginning –and may be huge–, this is assessed by using a MapReduce approach for optimization reasons.

Algorithm 1 Nodes Counter Mapper

```
1: procedure MAP(PageId id, Page p)
2:     if p is not empty then
3:         EMIT(uniqueKey, 1)
```

Algorithm 2 Nodes Counter Reducer

```
1: procedure REDUCE(Key k, Values $[v_1, v_2, \dots]$)
2:     for all value in values do
3:         $sum \leftarrow sum + value$
4:     EMIT(N, sum)
```

1.3 Second Phase: Graph Construction

In this phase we parse the information in the input file taking the information that interests us, i.e. the title of the page and the outgoing edges. Also, to each page, we provide the initial PageRank thanks to the already calculated total number of nodes.

Algorithm 3 Graph Construction Mapper

```
1: procedure MAP(PageId id, Page p)
2:     $title \leftarrow getTitle(p)$
3:     $outgoingEdges \leftarrow getOutgoingEdges(p)$
4:     EMIT(title, outgoingEdges)
```

Algorithm 4 Graph Construction Reducer

```
procedure InitReduce(Configuration c)
    $N \leftarrow c.numberOfNodes$
procedure Reduce(Title t, ListOfListOfEdges $[e_1, e_2, \dots]$)
    $initialPageRank \leftarrow \frac{1}{N}$
    $edges \leftarrow e_1$
    EMIT(title, {initialPageRank, edges})
```

In *Algorithm 4* only e_1 is considered because the title is a unique identifier of the pages, so the list of input values will always consist of a single element. Only one mapper will manage one page.

1.4 Third Phase: PageRank Estimation

In this section, the relaxed pagerank iteration is presented. The number of iteration is fixed at the start of the execution. We do not converge to a (or a more or less) consistent state, because the presence of dangling nodes will cause importance (i.e., pagerank mass) to leak out.

Algorithm 5 PageRank Computation Mapper

```
procedure Map(Key k, FormattedPage p)
    EMIT(p.title, {0, p.outgoingEdges})
    for all outgoingEdge in p.outgoingEdges do
        EMIT(outgoingEdge, $\{\frac{p.pagerank}{p.outgoingEdges.length}, NULL\}$)
```

Note: *outgoingEdge* is a title itself.

Algorithm 6 PageRank Computation Reducer

```
procedure InitReduce(Configuration c)
    $N \leftarrow c.numberOfNodes$
    $D \leftarrow Damping$
procedure Reduce(Title t, Nodes $[n_1, n_2, \dots]$)
    n new Node
    for all node in nodes do
        if node.hasOutgoingEdges() then
            n.outgoingEdges = node.outgoingEdges
        else
            $s = s + node.pagerank$
    $n.pagerank = \frac{(1-D)}{N} + D * s$
    EMIT(t, n)
```

1.5 Fourth Phase: Sorting

The final step is sorting the webpages by decreasing rank, this is done making advantage of the sorting mechanism of MapReduce.

Algorithm 7 Sorting Mapper

```
procedure Map(Key k, FormattedPage p)
    $title \leftarrow p.title$
    $pagerank \leftarrow p.pagerank$k
    EMIT(pagerank, title)
```

Algorithm 8 Sorting Reducer

```
procedure REDUCE(Pagerank rank, Titles [$t_1, t_2, \ldots$])
    for all title in titles do
        EMIT(title, rank)
```

2 — Hadoop Implementation

2.1 Introduction

In this section is briefly described some features done in the code for speeding up the execution of the code introduced in the pseudocode presented in the previous chapter. After that, the performance evaluation of the computation is reported.

2.2 First Phase: Nodes Counting

The counting is done making use of a combiner with the same code of the reducer, this helps up to speed up the processing at the reducer with a simple sum of three values (i.e., equal to the number of mappers used). The value computed is stored in a temporary file that we will erased after being read and stored inside the *Configuration* object.

2.3 Second Phase: Graph construction

Parsing is a job that doesn't not need to create a conglomerate value, but it creates records which, each of them, is treated as a standalone piece of information, thus using multiple reducers is possible. We decide to use three reducers since we have three working nodes.

2.4 Third Phase: PageRank Estimation

As in the *Graph construction* phase multiple reducers are used for accelerating the computation. Thus, from the first iteration to the last one this MapReduce computation takes multiple inputs and produces multiple outputs.

2.5 Forth Phase: Sorting

Since the previous phase constructs multiple outputs, the sorting phase has to take multiple inputs. Here a single reducer is used in order to take advantage of the automatic sorting done over the keys by Hadoop. Keys are in this case the pagerank value of each page, and are passed by the mapper to the reducer as **DoubleWritable** objects.

2.6 Performance

For measuring the performance we considered three files with a different number of nodes:

- **wiki-micro.txt**: number of nodes 2427.
- **dataset5.txt**: number of nodes 5000. This dataset is synthetic dataset created using the same structure of *wiki-micro.txt*. Each node has a random value of outgoing edges between 0 and 10.
- **dataset10.txt**: number of nodes 10000. This dataset is synthetic dataset created using the same structure of *wiki-micro.txt*. Each node has a random value of outgoing edges between 0 and 10.

and we test them over different numbers of iteration: 5, 10, 15.

Table 1: Performance

File	N Iter 5	N Iter 10	N Iter 15
wiki-micro.txt	185557	3181196	432149
dataset5.txt	-	-	-
dataset10.txt	-	-	-

The values in **Table 1** are all in milliseconds.

3 — Spark Implementation in Java

4 — Spark Implementation in Python