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

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

Algorithm 2 Nodes Counter Reducer

```
procedure REDUCE(Key k, Values [v1, v2, ...])
    for all value in values do
        sum ← sum + value
    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

```
procedure MAP(PageId id, Page p)
    title ← getTitle(p)
    outgoingEdges ← getOutgoingEdges(p)
    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 $[n_1, n_2, \dots]$)
    if title = DANGLING **then**
        for all node **in** nodes **do**
            $s \leftarrow s + node.pagerank$
        EMIT(t, {s, NULL})
    else
        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 $[t_1, t_2, \dots]$)
  for all title **in** titles **do**
    EMIT(title, rank)

2 — Hadoop Implementation

3 — Spark Implementation in Java

4 — Spark Implementation in Python