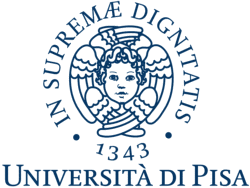
a.a. 2020-2021



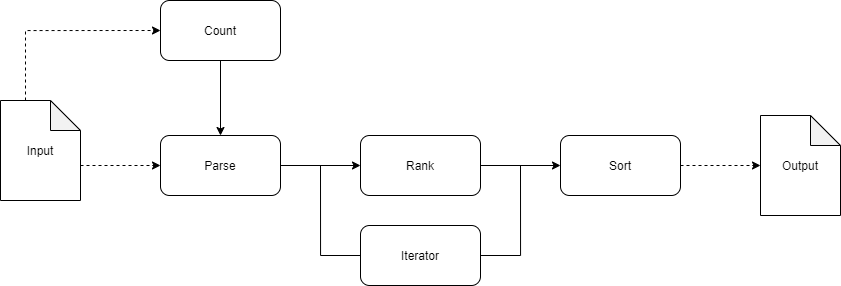
CLOUD COMPUTING

THE PAGE RANK ALGORITHM IN MAP REDUCE

Presented by:   
Dini federica  
PANICHI NICCOLò  
BICCHIERINI IACOPO  
BIANCHI LORENZO

## Algorithm design

In order to implement the PageRank algorithm we divided the computation in different stages:



In the Count stage the number of pages are counted.

In the Parse stage the nodes informations are retrieved from the xml pages, so node objects with title, adjacency list and page rank (initially set as 1 / TotalPages) are generated.

In the Rank stage the pages distribute their PageRank mass to through their out-links, and the PageRank values are updated. This stage is repeated for a number of times specified from the command line thanks to the Iterator.

In the Sort stage the Page are sorted in ascending order of PageRank.

The output directory is structured as following:

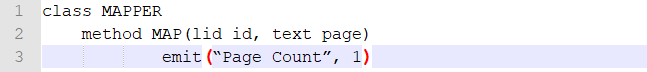
Immagine che contiene testo

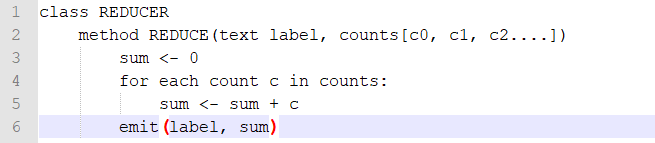
Descrizione generata automaticamente

Each stage reads the output of the precedent stage. The output of the Sort stage is the final result.

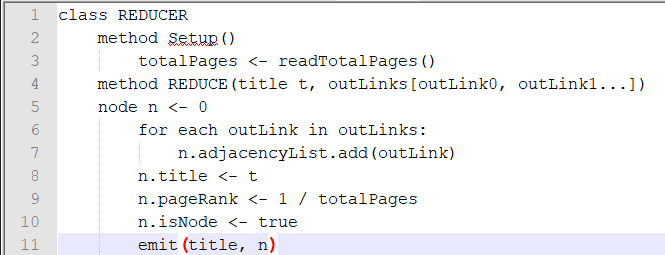
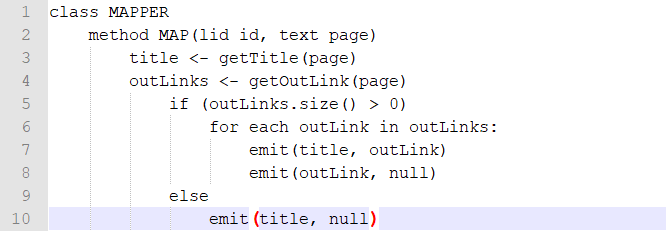
## PseudoCode

**Count stage**

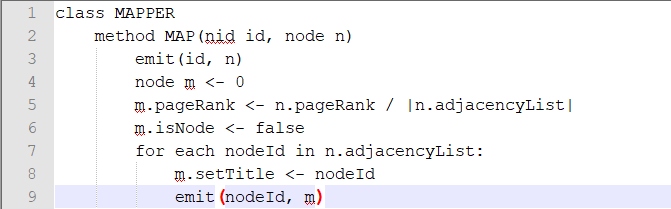
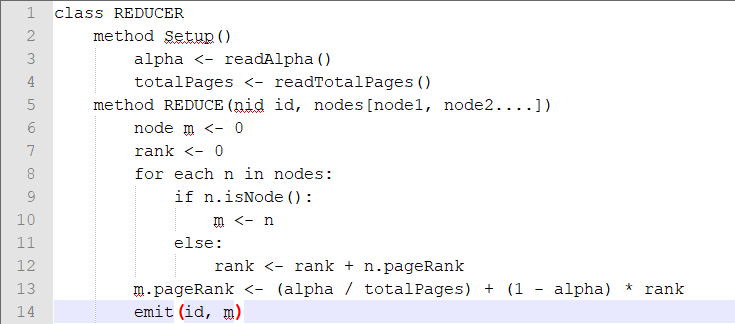
****

****

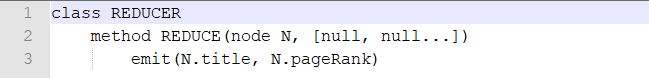
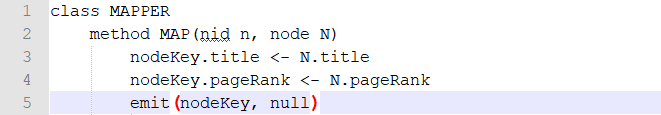
**Parse stage**

****

**Rank stage**

****

**Sort stage**

****

## Efficiency Issues

On the Hadoop implementation, we used customized object Node: a WritableComparable implementation which stores all the information of a node (title, adjacency list and page rank) used as value by Parse Reducer, Rank Mapper and Rank Reducer, and as key by the SortMapper and SortReducer. This object is used to pass nodes informations along the stage pipeline, to distribute the page rank mass along the graph and, through the implementation of the compareTo method, to sort results directly exploiting the Shuffle & Sort phase of the framework.

To reduce the quantity of intermediate data produced, we implemented some Combiners and tested them on the wiki-micro dataset:

* An In-Mapper Combining in the Count stage(from 2427 to 1 intermediate pair)
* A Combiner in the Rank stage(from 80592 to 68966 intermediate pair)

They allow us to reduce the amount of data moved across the network.

We exploited the setup method to read configuration files in order to get parameters and to read the output of the Count stage. The cleanup method is used by the CountMapper in order to emit the total page count.

We made some test with our application using more than 1 Reducer, that is passed from command line to *Parse* and *Rank* class. We set 3 iteration in the Rank and alpha 0.8 in all the tests.

In order to analyze the performance varying with the number of reducers, we test our application with 1, 2, 3, 5 Reducer. The performance will be analyzed using this 4 values:

* Map Task Failed
* Total time spent by all map tasks (ms)
* Total time spent by all reduce tasks (ms)

**NUMBER REDUCER** : 1

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Map Task Failed** | **Maps Time (ms)** | **Reduce Time (ms)** |
| **Parse** | 0 | 2595 | 4570 |
| **Rank1** | 1 | 8107 | 3934 |
| **Rank2** | 1 | 9388 | 4085 |
| **Rank3** | 3 | 20073 | 6053 |
| **Sort** | 1 | 7595 | 2420 |

**NUMBER REDUCERS**: 2

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Map Task Failed** | **Maps Time (ms)** | **Reduce Time (ms)** |
| **Parse** | 1 | 7053 | 8364 |
| **Rank1** | 1 | 153446 | 14511 |
| **Rank2** | 1 | 8908 | 8188 |
| **Rank3** | 1 | 7324 | 7016 |
| **Sort** | 0 | 6561 | 3798 |

**NUMBER REDUCERS** : 3

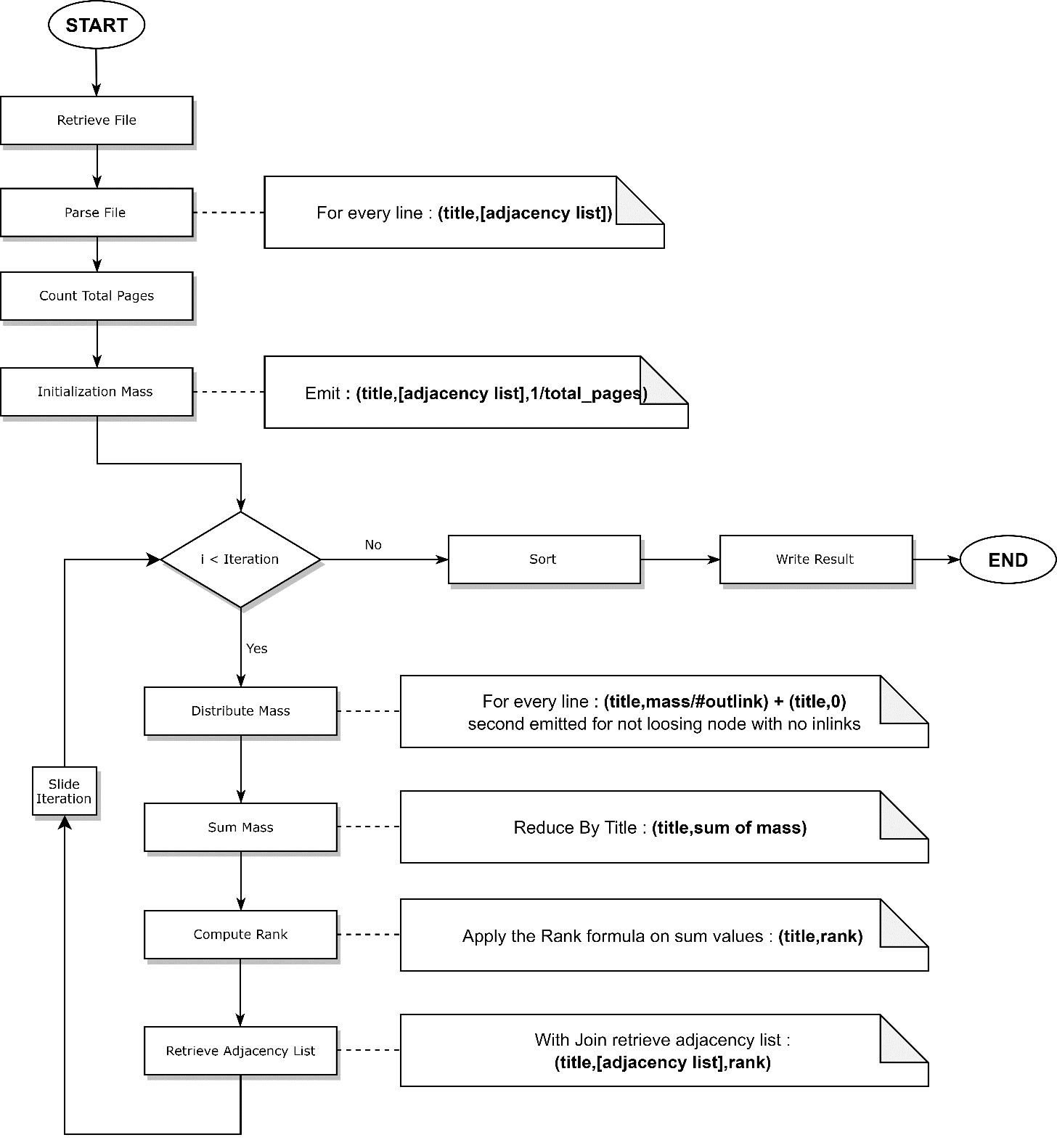
|  |  |  |  |
| --- | --- | --- | --- |
|  | **Map Task Failed** | **Maps Time (ms)** | **Reduce Time (ms)** |
| **Parse** | 0 | 2807 | 9847 |
| **Rank1** | 1 | 16024 | 11622 |
| **Rank2** | 6 | 76442 | 79245 |
| **Rank3** | 1 | 25494 | 11973 |
| **Sort** | 3 | 34603 | 3785 |

**NUMBER REDUCERS** : 5

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Map Task Failed** | **Maps Time (ms)** | **Reduce Time (ms)** |
| **Parse** | 0 | 2739 | 19299 |
| **Rank1** | 8 | 117792 | 114711 |
| **Rank2** | 8 | 116096 | 136699 |
| **Rank3** | 4 | 89197 | 71278 |
| **Sort** | 2 | 78695 | 12004 |

As we can see from these tables of data, increasing the number of reducers make increase the ranking and sorting operation time. The meaning of this result is that our dataset is very tiny and when we increase the number of reducer, we introduce more I/O operation because the data need to be slit across the reducers, that will require network transfer time. So instead of creating 1 file for the reducer, it will creates a lot of files.

## Spark implementation



**RDD Persistence**

We have decided to store in main memory **node RDD** using cache() function because is the most used RDD and it not changes during the execution. This RDD is used to count the pages, to initiate the mass and at every iteration to restore the tuple with the adjacency list.