Università
Ca'Foscari
Venezia

LEARNING WITH MASSIVE DATA
CM 0622-1

ASSIGNMENT 3

Student:

Giosuè Zannini 873810

Academic year 2022/2023

Introduction

In this third assignment I had to develop an algorithm (I used Python as programming language) that given a collection of documents determines all pairs similarity (cosine similarity) that exceed a determined threshold. As output of algorithm I decided to give a dictionary where the keys are all docs that with another doc outdo the threshold and as values a list of pairs composed by other doc and the cosine similarity between them. For this assignment I develop four algorithms:

- Sequential implementation.
- Parallel implementation using Apache Spark.
- Parallel implementation using Apache Spark that solve problem 1 seen in pdf 23 of the course.
- Parallel implementation using Apache Spark that solve problem 1 and 2 seen in pdf 23 of the course.

About dataset being that the sequential implementation takes a lot of time to compute I decided to reduce in random way the dimension of all datasets, I used the following taken from BeIR:

Dataset	# Docs	# Sub docs
NFCorpus	3'600	240
SciFact	5'000	120
Quora	523'000	600

Moreover being that each subset has too few documents to reach a high threshold I implemented an algorithm to increase the similarity between docs, the idea is given a dataset, copy k docs and remove some of their contents determined by a percentage, return the original dataset with appended these k docs.
The hardware used to perform this task is as follows:

- Processor: AMD Ryzen 5 3500U with Radeon Vega Gfx (8CPUs) $\sim$2.1GHz.
- RAM: 8GB.

Idea implementation sequential algorithm

Given the CSR matrix of TF-IDF calculated from sklearn I decided to transform this matrix in a list of pairs where the first element is the doc id and the second element is the TF-IDF (using CSR compression) vector associated and from it I implement an optimization which uses the idea that given a TF-IDF of a document i find the minimum number of terms that another document must have different from zero to outdo the threshold (to find this number I ordered the document i in descendent way and compute $d_i[j]^{\mathrm{T}} d_i[j]$ till the term j where it identifies the minimum number of terms to outdo the threshold), this idea united with the fact that the docs are sorted in descendent way considering as ordering the number of terms different from zero gives me the possibility to skip some similarity computations. Another optimization that I decided to implement regards the fact that the computation of similarity between documents gives a symmetric matrix of similarity, for this reason I decided to compute only the upper bound of the matrix.

Idea implementation parallel algorithm

As in the previous section I started again from the list of pairs and being that the purpose is using Spark I followed this path to perform the algorithm:

- **map** function transforms input in $[(\text{term\_id}_1, (\text{doc\_id}_1, \text{TF-IDF}_1)), \dots, (\text{term\_id}_m, (\text{doc\_id}_n, \text{TF-IDF}_n)]$.
- **groupByKey** function transforms input in $[(\text{term\_id}_1, [\dots, (\text{doc\_id}_i, \text{TF-IDF}_i), \dots]), \dots, (\text{term\_id}_m, [\dots, (\text{doc\_id}_j, \text{TF-IDF}_j), \dots])]$.
- **map** function (compute cosine) transforms input in $[(\text{term\_id}_1, [\dots, (\text{doc\_id}_i, \text{doc\_id}_j, \cos_{i,j}), \dots]), \dots, (\text{term\_id}_m, [\dots, (\text{doc\_id}_k, \text{doc\_id}_l, \cos_{k,l}), \dots])]$.
- **values** function transforms input in $[[\dots, (\text{doc\_id}_i, \text{doc\_id}_j, \cos_{i,j}), \dots], \dots, [\dots, (\text{doc\_id}_k, \text{doc\_id}_l, \cos_{k,l}), \dots]]$.
- **flatMap** function transforms input in $[\dots, (\text{doc\_id}_i, \text{doc\_id}_j, \cos_{i,j}), \dots, (\text{doc\_id}_k, \text{doc\_id}_l, \cos_{k,l}), \dots]$.
- **distinct** function transforms input removing duplicates.
- **map** function transforms input in $[\dots, (\text{doc\_id}_i, (\text{doc\_id}_j, \cos_{i,j})), \dots]$.
- **groupByKey** function transforms input in $[(\text{doc\_id}_i, [\dots, (\text{doc\_id}_j, \cos_{i,j}), \dots]), \dots, (\text{doc\_id}_k, [\dots, (\text{doc\_id}_l, \cos_{k,l}), \dots])]$.
- **collect** function to return result to driver program.

To solve problem 1 and 2 I used the algorithm explained by professor during the lesson.

Performance discussion

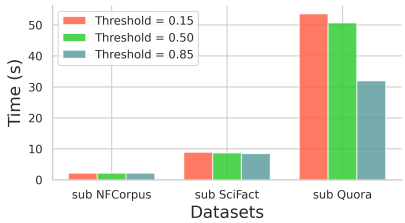
In this section I want to check the difference between sequential and parallel implementation (considering only the time to compute the similarity, i.e. removing precomputation time) and discover how algorithms using Spark change in performance by changing:

- Workers, I used as values 1, 2, 4, 8.
- Threshold, I used as values 0.15, 0.50, 0.85.
- Datasets.

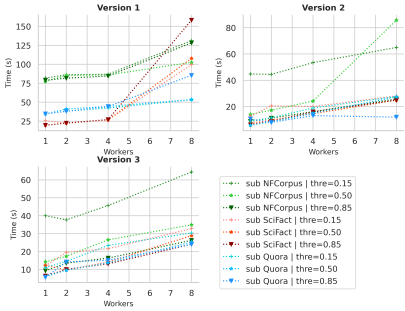
First of all I would like to emphasize the time difference between sequential algorithm and parallel algorithm in this table:

Dataset	Min seq. time among thresholds (s)	Min parall. time among thresholds, workers and versions (s)
sub SciFact	2.130	6.307
sub NFCorpus	8.496	8.843
sub Quora	31.997	5.729

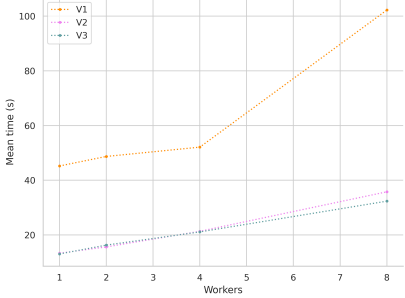
From it I can see that with the size increase the Spark version achieves a better result than the sequential version. Moreover I want to show how the optimizations implemented in sequential algorithm help to reduce execution time as the threshold value increases.



Now I want to show the plot regarding the Spark implementations:



From these plots I can notice that the behavior to the increase of the workers turns out the same for all the versions of the algorithm, in fact to increasing of the workers the times of execution dilate, this is a strange behavior in my opinion being that I thought that more workers implied better performance. Then I reasoned that I probably have this behavior because in this algorithm there is a lot of synchronization and to verify my thesis I tried to run the pi approximation example algorithm (which has very little synchronization) but the behavior remained the same, so probably the problem is not synchronization but something else. Furthermore I notice that the subset of Quora has the best results, I think this is happened because the number of terms in the TF-IDF vector is the smallest (sub Quora terms: 1895, sub SciFact terms: 4407, sub NFCorpus terms: 5896). Another thing I want to mention is that the variance between the times is less in the second and third versions and that in these two variations of the algorithm when the threshold increases the times decrease, since the number of pairs (term_id, tf_idf) to be analyzed become lower. This does not happen in the basic version as the calculation of the similarity remains the same as the threshold changes.
In the plot below I want to verify which algorithm among three that I develop with Spark is the fastest (where for each algorithm I considered the mean between datasets and thresholds for determining time):



From it I notice that the algorithm with the lowest times is V3, but even V2 is very close. So the optimizations applied to solve problem 1 and problem 2 have decreased execution time consistently.