Learning Over Massive Data

Assignement 2 – SimHash

Davide Vitagliano 904120

Academic Year 2023/2024

1. Execution

```
spark-class org.apache.spark.deploy.master.Master -h localhost
```

```
spark-class org.apache.spark.deploy.worker.Worker spark://localhost:7077 -c 2
```

```
spark-submit –master spark://localhost:7077 –py-files doc_manipulation.py –executor-memory 12g main.py [params] | tee ./results/[file_name].txt
```

2. Algorithm

To solve the task of finding near-duplicate documents, we implemented a MapReduce algorithm in pyspark using the functional programming paradigm. The process computes SimHash to approximate the similarity between documents, and it is divided into two phases.

Map phase: it receives a collection of documents, and it transforms it into a matrix of tf-idf value of each term in the document. Then, to compute the signature of each document, we randomly generate a vector of -1 and 1 for each term in the document, we compute the dot product between the term vector and the document vector and we normalize it and we divide the into pieces. Finally, we apply a map function to the collection of signatures to emit (docID, piece) pairs.

Reduce phase: it joins each pair with the entire signature of the document, then it groups by piece to get all the documents that have a piece in common. Moreover, it applies a map function that, for each pair of documents, counts the number of pieces that are equal, and it computes the hamming distance between the two signatures only if the number of shared pieces is greater than or equal to 1 but less than the total number of pieces. In the latter case, the two documents are equal, so we count them as duplicates. Finally, we compute the actual similarity between the two signatures, and if it is greater than the thrashold we emit (docID1, docID2, similarity) pairs.

The mapper function's space complexity is determined by the size of the simhash_pieces RDD, which is proportional to the number of documents. Therefore, the space complexity is **O(docs).**

The reducer function's space complexity is determined by the size of the grouped RDD. In the worst case, every document shares a piece with every other document, so the space complexity is **O(docs^2).** However, in practice, it's likely to be much less than this because not all documents will share a piece.

The mapper function's time complexity is determined by the computation of the SimHash, which is are linear with respect to the number of documents and the number of terms, so the time complexity is **O(docs*terms).**

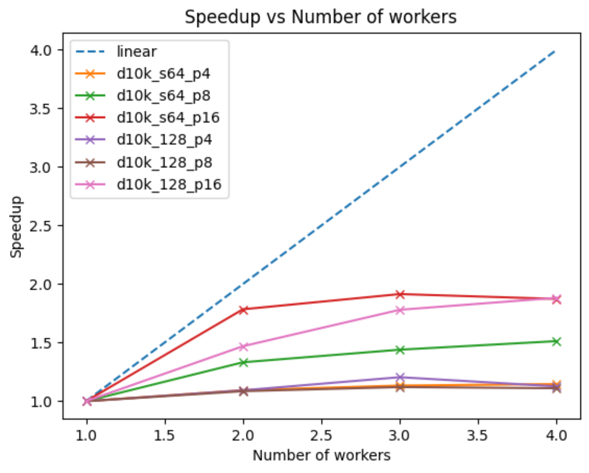
The reducer function's time complexity is determined by the grouping operation and the nested loop that computes the cosine similarity for each pair of documents. The grouping operation is linear with respect to the number of documents, and the nested loop is quadratic with respect to the number of documents that share a piece. Therefore, the time complexity is **O(docs + k^2)**, where k is the maximum number of documents that share a piece.

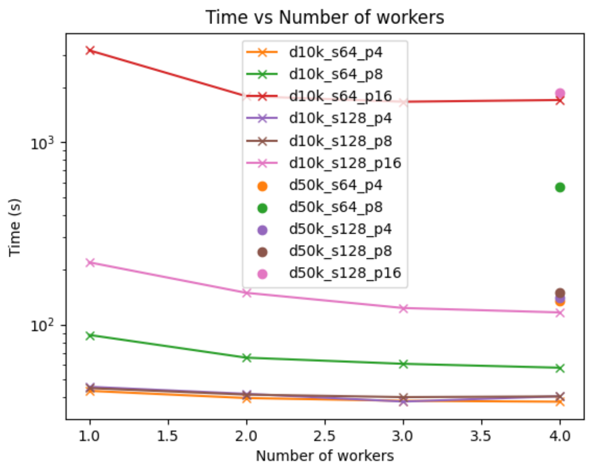
3. Testing

The performance and scalability of the algorithm is evaluated by measuring the speedup of the version with more workers with respect to the single worker version; every worker has 2 cores. The execution time is measured with perf_counter() function of the time library. For all testings has been used a similarity threshold of 95%; different results on the number fo pairs, at parity of signature and pieces, are due to the randomness of the algorithm when generating the term vectors.

Workers	documents	signature	pieces	mapper time	reducer time	total time	similar pairs	duplicates
4	10000	64	4	31.86	5.91	37.77	308	421
4	10000	64	8	33.89	24.11	58.00	1785	947
4	10000	64	16	33.51	1670.63	1704.14	7360	10130
4	10000	128	4	34.54	5.93	40.47	368	1154
4	10000	128	8	34.53	5.85	40.38	836	721
4	10000	128	16	36.15	80.44	116.59	1495	760
4	50000	64	4	114.55	20.32	134.87	9965	22692
4	50000	64	8	119.68	450.75	570.43	31364	15167
4	50000	128	4	119.43	20.66	140.09	4917	21007
4	50000	128	8	117.68	31.82	149.50	11650	22162
4	50000	128	16	117.16	1755.31	1872.47	48654	22381

From the chart and table, we observe is that the number of pairs increases as the number of pieces in the signature increases. This is because the more pieces there are, the more likely it is that two documents will share a piece and be grouped together. Another strange fact is that the number of pairs found with different number of workers, but at parity of signature and pieces is different, this could be because the data isn't evenly distributed among the workers. We can also see that the speedup is sublinear because the overhead of coordinating the workers starts to outweigh the benefits of parallelism. The speedup is also affected by the number of pieces in the signature, with more pieces leading to a higher speedup.


Speedup vs Number of workers
Speedup
Number of workers
linear
d10k_s64_p4
d10k_s64_p8
d10k_s64_p16
d10k_128_p4
d10k_128_p8
d10k_128_p16


Time vs Number of workers
Time (s)
Number of workers
d10k_s64_p4
d10k_s64_p8
d10k_s64_p16
d10k_s128_p4
d10k_s128_p8
d10k_s128_p16
d50k_s64_p4
d50k_s64_p8
d50k_s128_p4
d50k_s128_p8
d50k_s128_p16