CSMR: A Scalable Algorithm for Text Clustering with Cosine Similarity and MapReduce

Giannakouris-Salalidis Victor, Plerou Antonia, Sioutas Spyros

CSMR: A Scalable Algorithm for Text Clustering with Cosine Similarity and MapReduce

Giannakouris - Salalidis Victor, Plerou Antonia, Sioutas Spyros

Ionian University, Department of Informatics, Greece
{p12gian1,tplerou,sioutas}@ionio.gr

Abstract. As Internet develops rapidly huge amounts of texts need to be processed in a short time. This entails the necessity of fast, scalable methods for text processing. In this paper a method for pairwise text similarity on massive data-sets, using the Cosine Similarity metric and the tf-idf (Term Frequency-Inverse Document Frequency) normalization method is proposed. The research approach is mainly focused on the MapReduce paradigm, a model for processing large data-sets in parallel manner, with a distributed algorithm on computer clusters. Through MapReduce model application on each step of the proposed method, text processing speed and scalability is enhanced in reference to other traditional methods. The CSMR (Cosine Similarity with MapReduce) method's implementation is currently at the implementation stage. Precise and analytical conclusions concerning the efficiency of the proposed method are to be reached upon completion and review of the overall project phases.

Keywords: MapReduce, Hadoop, TF-IDF, Text Mining, Cosine Similarity

1 Introduction

Nowadays, as the data amount grows rapidly, challenge of big data need to be faced [1] in various domains such as Business Intelligence [2] or Bioinformatics [3, 4]. With ever increasing volume of text documents, the abundant texts flowing over the Internet, huge collections of documents in digital libraries and digitized personal information are collected quickly every day [5]. In this paper, an innovative method for text similarity measuring with the use of common techniques and metrics is proposed. In particular, a prospective of applying tf-idf [6] and Cosine Similarity [7] measurements on distributed text processing is further analyzed. The CSMR (Cosine Similarity with MapReduce) method includes the component of document pairwise similarity calculation. Especially, CSMR method performs pairwise text similarity with the use of a parallel and distributed algorithm which scales up, regardless the massive input size. This is utilized with the use of MapReduce component of the Hadoop Framework. The authors' proposed method consists by two main components: tf-idf and Cosine Similarity. In this study, these components are designed by following the concept of the MapReduce programming model. Initially, the terms of each document are

counted. Secondly, texts are normalized with the use of tf-idf. Finally, Cosine Similarity of each document pair is calculated and results are given as an output. The CSMR method is proposed as a faster and more efficient method comparing to the traditional methods. This is due to MapReduce model implementation in each algorithmic step tends to enhance method's efficiency as well as to the aforementioned techniques innovative blend.

2 Related Work

There are quite many cases where several methods have been used for measuring similarity among texts.

Tamer Elsayed et.al [8] method focuses on a MapReduce algorithm for computing pairwise document similarity in large document collections. The algorithm proposed exhibits linear growth in running time and space, in terms of the number of documents. This algorithm is suggested as an example of a programming paradigm that could be useful for a broad range of text analysis problem. Another approach has been proposed by Bin Li et.al [9], i.e. a tf-idf algorithm based on the Hadoop framework. This method is using the MapReduce model provided by Hadoop in order to improve the efficiency of traditional tf-idf algorithm. This case study showed that in the case of massive data computing, Hadoop framework implementation is more efficient comparing to the traditional method.

Jacob Bank et.al [10] use a different approach in order to analyze the vast amounts of data associated with large-scale social networks on the web with the use of the MapReduce program. The Jaccard similarity coefficient between users of Wikipedia based on co-occurrence of page edits is proposed. After several separate linear time computations it was con-firmed that this approach was superior to quadratic computations on long lists of data. Calculating the Jaccard Similarity Coefficient with Map Reduce for Entity Pairs in Wikipedia.

Furthermore, Jian Wan et.al [11] proposed an approach about how document clustering for large collection could be efficiently implemented with MapReduce. Additionally tf-idf and K-Means algorithm on MapReduce design and implementation is described in order to improve algorithm efficiency and effectiveness. Experimentation confirmed the scalability of processing mass data proposed method.

Ping Zhou et.al [12] supplementary research in reference to large-scale data sets clustering amplification a parallel K-Means algorithm based on MapReduce framework is proposed. Model's implementation results illustrated that the proposed clustering algorithm running on Hadoop cluster preserve a higher performance while handling large-scale document automatic classification. In the above mentioned methods dealing with text clustering, there is none or only a slight and indirect approach via Cosine Similarity in order to improve processing speed and scalability.

Finally, according to Rada Mihalcea et.al [13] approach a method for measuring the semantic similarity of short texts, using corpus-based and knowledge-based measures of similarity is presented. Through experiments per-formed on a paraphrase data set, semantic similarity method outer-forms methods based on simple lexical matching,

resulting in up to 13% error rate reduction with respect to the traditional vector-based similarity metric. On the contrary, they focus to the aspect that Cosine Similarity, tf-idf as well as other methods can be used for text similarity measuring. There are also some approaches using MapReduce but, according to authors' knowledge, none of them proposes a model with tf-idf and Cosine function.
Authors' proposed method combines overall of these 3 powerful techniques, i.e. tf-idf, Cosine Similarity and MapReduce and provides a powerful and scalable algorithm suitable for various purposes on Data Mining, especially on Text Processing on big, massive data-sets.

3 Basic Background

According to the project needs, three techniques had been chosen: The Vector Space Model, tf-idf and Cosine Similarity. Each of these techniques is being described in detail below.

3.1 Vector Space Model

Vector Space Model is an algebraic model for representing text documents as vectors. [14] With the use of this model, each term of a document and each number of occurrences in the document could be represented [15]. For instance, the document *d1 = "This is a vector, this is algebra"* based on a vocabulary $V(t)$ could be represented as follows:

$$V(t) = \begin{Bmatrix} 1, t = "this" \\ 2, t = "is" \\ 3, t = "a" \\ 4, t = "vector" \\ 5, t = "algebra" \end{Bmatrix}$$

$$d1 = \left(tf(1,d1), tf(2,d1), tf(3,d1), tf(4,d1), tf(5,d1)\right)$$

$$= (2,2,1,1,1)$$

Where *d1* is the document and $tf(t,d_i)$ is the term frequency of the t-term in the i^{th} document.

3.2 Tf-Idf

In Text Mining, tf-idf (Term Frequency-Inverse Document Frequency) [6] is a numerical statistic that reflects the significance of a term in a document in a corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. Tf-idf algorithm is usually used in search engine, web data mining, text similarity computation and other

applications [16]. These applications are often faced with massive data processing. According to Bin Li [9]approach the tf-idf of a term is calculated with the use of the following formula:

$$TF \times IDF = \frac{n_{i,j}}{|\, t \in d_j \,|} \times \log \frac{|\, D \,|}{|\, d \in D : t \in d \,|}$$

3.3 Cosine Similarity

Cosine Similarity is a measure of similarity between two vectors of an inner product space that measures the cosine of the angle between them [17]. For document clustering, there are different similarity measures available. The Cosine function is proposed as the most commonly used. For two documents A and B, the similarity between them is calculated with the use of the following formula:

$$\cos(\mathrm{A}, \mathrm{B}) = \frac{A \cdot B}{\| \mathrm{A} \| \| \mathrm{B} \|} = \sum_{i=1}^{n} \frac{A_i \times B_i}{\sqrt{\sum_{i=1}^{1} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

When the cosine value is computed to be 1, that indicates that the two documents are identical and while it is computed to be 0 if there is nothing in common between them (i.e., their document vectors are orthogonal to each other). The attribute vectors A and B are usually the term frequency vectors of the documents.

3.4 Hadoop & MapReduce

Hadoop software library [18], is a framework developed by Apache, suitable for scalable, distributed computing. It allows storage and large-scale data processing across clusters of commodity servers [19]. The innovative aspect of Hadoop is that there is no absolute necessity of expensive, high-end hardware. Instead, it enables distributed parallel processing of massive amounts of data [20] on industry-standard servers with high scalability for both data storing and processing. Therefore it is considered to be one of the most popular frameworks for Big Data Analytics. Especially, Hadoop has two main subprojects: HDFS (Hadoop Distributes File System) & MapReduce.
MapReduce [21] is the main component of Hadoop. It's a programming model that allows massive data processing across thousands of servers in a Hadoop cluster. The MapReduce paradigm is derived from the Map and Reduce functions of the Functional Programming model [22]. A MapReduce program constitutes from the Mappers and the Reducers. In the Map phase, the master node the master node divides the input into smaller partitions and distributes them to the worker nodes. Then a worker node may repeat the same step recursively. As soon as this procedure is completed, the master node collects the key-value pairs resulted from the Mappers and distributes them to the Combiners to combine the pairs with the same key. This phase is known

as the Shuffle & Sort phase. Finally, the key-value pairs are distributed to the Reducers that produce the final output. This step is called the Reduce phase. MapReduce program procedure is visualized as follows:

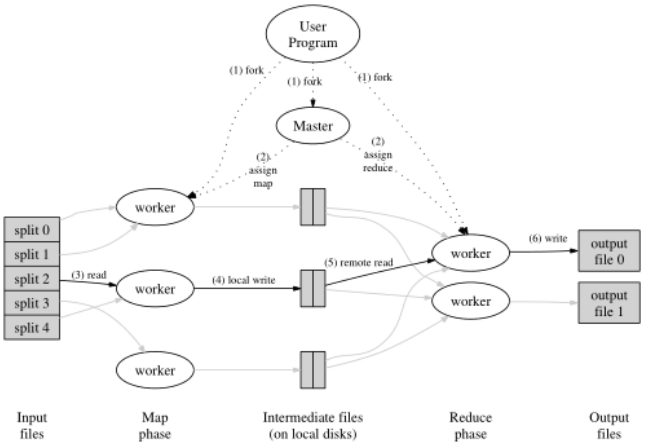

Fig. 1. MapReduce Procedure Visualization

4 Method

4.1 Description

Authors' proposed method for measuring text similarity applying MapReduce consists of 4 stages. At the first stage, occurrences of each term in our documents are counted. Then, the term frequency of every one term in each document is measured. Thereafter the tf-idf of each term is measured and finally the cosines of the pairs are calculated in order to estimate the similarity among them. MapReduce model was used in order to design each one of the above mentioned steps. The algorithm paradigm in pseudocode and further analysis of each step is disposed in details in the next section.

4.2 MapReduce Stages

In the 1st implementation stage the occurrences of each term in every document are counted. The algorithm applied is as follows:

Algorithm 1*: Word Count*

```
1:  class Mapper
2:    method Map( document )
3:      for each term ∈ document
4:        write ( ( term , docId ) , 1 )
```

```
5:
6:   class Reducer
7:     method Reduce( ( term , docId ) , ones[ 1 , 1 , … , n ] )
8:       sum = 0
9:       for each one ∈ ones do
10:        sum = sum +1
11:      return ( ( term , docId ) , o )
12:
13:    /* { o ∈ N : the number of occurrences } */
```

Initially, each document is divided into key-value pairs. The term is selected as the key as well as the number one as the value. That is denoted as (term, 1) where key corresponds to the term and the value to the number one respectively. This phase is known as the Map Phase. In the Reduce Phase each pair is taken and the sum of the list of ones for the term is computed. Finalizing, the key is set as the tuple (document, term) and the value as the number of occurrences respectively.

In the 2nd implementation phase the overall number of terms of each document is computed.

***Algorithm 2**: Term Frequency*

```
1:   class Mapper
2:     method Map( ( term , docId ) , o )
3:       for each element ∈ ( term , docId )
4:         write ( docId, ( term, o ) )
5:
6:   class Reducer
7:     method Reduce( docId, (term, o) )
8:       N = 0
9:       for each tuple ∈ ( term, o ) do
10:        N = N + o
11:      return ( (docId, N), (term, o) )
```

By this algorithm implementation, concerning the Map Phase, the input is divided into key-value pairs while the *docId* is set as the key in addition to the tuple *(term, o)* as the value. In the reduce phase the total of terms in each document is counted and the key-value pairs are returned with the (*DocId, N)* as the key as well as the tuples *(term, o)* as the value (*N* is the total of terms in the document). The key-value pairs are returned with the tuples (docId, N) as the key and the tuples (term, o) as the value, where N is the total of terms in the document.

In the 3rd implementation stage the tf-idf of each term in a document is computed with the use of the following formula:

$$tfidf = \frac{n}{N} \frac{|D|}{|\{d \in D : t \in d|}$$

Where |D| is the number of the documents in corpus and $|\{d \in D : t \in d\}|$ number of documents where t-term appears.

***Algorithm 3**: Tf-Idf*

```
1:  class Mapper
2:    method Map( ( docId , N ), ( term , o  ) )
3:      for each element ∈ ( term , o  )
4:        write ( term, ( docId, o, N ) )
5:
6:  class Reducer
7:    method Reduce( term, ( docId , o , N ) )
8:      n = 0
9:      for each element ∈ ( docId , o , N ) do
10:       n = n + 1
11:     tf = o / N
12:     idf = log(| D | /(1+n))
13:     return ( docId, ( term , tf×idf  ) )
14:
15:   /* Where |D| is the number of documents in the corpus */
```

Applying the aforementioned algorithm, during the Map Phase the term is set as the key as well as the tuple *(docId, o, N)* as the value. In that case, the number of documents is calculated by the reducer, where the term appears and the result to the *n* variable is set. The term frequency is subsequently calculated plus the inverse document frequency of each term as well. Finally, key-value pairs with the *docId* as the key and the tuple *(term, tf×idf)* as the value are taken as a result.

In the 4th and final implementation phase all the possible combinations of two documents pairs are provided and cosine for each of them is computed. Assuming that there are *n* documents in the corpus, a similarity matrix of size is generated as follows:

$$\binom{n}{2} = \frac{n!}{2!(n-2)!}$$

Algorithm 4: Cosine Similarity

```
16: class Mapper
17:   method Map( docs )
18:     n = docs.length
19:
```

```
20:     for i = 0 to docs.length
21:       for j = i+1 to docs.length
22:         write ( ( docs[i].id, docs[j].id ),( docs[i].tfidf, docs[j].tfidf ) )
23:
24:  class Reducer
25:     method Reduce( ( docId_A, docId_B ),( docA.tfidf, docB.tfidf ) )
26:       A = docA.tfidf
27:       B = docB.tfidf
28:      cosine = sum( A×B  )/ (sqrt( sum(A^2) )× sqrt( sum(B^2) ))
29:     return ( (docId_A, docId_B), cosine )
```

In the Map phase, implementing the abovementioned algorithm, every potential combination of the input documents is generated and the document IDs for the key as well as the tf-idf vectors for the value is set. Within the Reduce phase, cosine for each document pair is calculated and the similarity matrix is also provided. Algorithm 4 is visualized as follows:

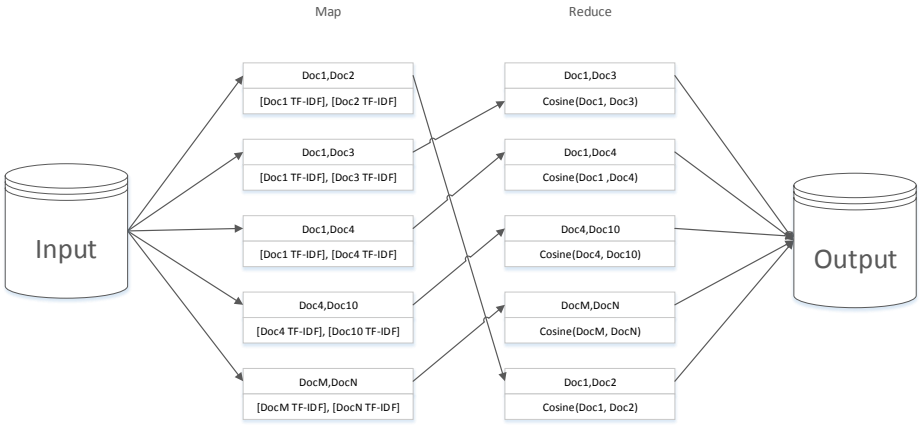

Fig. 2. Algorithm 4 Visualization

5 Discussion

In this case study, popular methods for measure the similarity of texts had been used. In particular, tf-idf and Cosine Similarity were adjusted with the MapReduce model in order to propose an innovative and scalable method. Authors' approach enhances the innovative aspect of the MapReduce programming paradigm in the field of text processing. The key contribution is that the proposed method enhances the procedure of measuring the text similarity with the Cosine metric and increase algorithm scalability. The implementation of the aforementioned techniques on computer clusters run-

ning the Hadoop Distributed File System (HDFS) blended with MapReduce also ensure algorithms effectiveness. The proposed method is currently at the design and implementation stage. Therefore, more clear and specific conclusions for its efficiency, as well as proposals for revisions and improvement, are to be provided after the projects' overall implementation.
Authors' future work concerns the finalized proposed method version as well as statistically analyzed results of the data collected during piloting implementation procedure presentation. In addition the software's design and development for the CSMR algorithm implementation on real text files is ongoing. An additional research approach is the implementation of the abovementioned algorithm with the use of tools like Apache Spark and Scala [23] as well as an Open Source project implemented in Java.

Acknowledgements. This research has been co-financed by the European Union (European Social Fund – ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: Thales. Investing in knowledge society through the European Social Fund.

References

1. Wu X, Zhu X, Member S (2014) Data Mining with Big Data. 26:97–107.

2. Chen H, Storey VC (2012) B USINESS I NTELLIGENCE AND A NALYTICS : F ROM B IG D ATA TO B IG I MPACT. 36:1165–1188.

3. Taylor RC (2010) An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics. BMC Bioinformatics 11 Suppl 1:S1. doi: 10.1186/1471-2105-11-S12-S1

4. Matsunaga A, Tsugawa M, Fortes J (2008) CloudBLAST: Combining MapReduce and Virtualization on Distributed Resources for Bioinformatics Applications. 2008 IEEE Fourth Int Conf eScience 222–229. doi: 10.1109/eScience.2008.62

5. Huang A (2008) Similarity Measures for Text Document Clustering.

6. Ramos J, Eden J, Edu R Using TF-IDF to Determine Word Relevance in Document Queries.

7. Tata S, Patel JM, Science C, Arbor A (2007) Estimating the Selectivity of tf-idf based Cosine Similarity Predicates. 36:7–12.

8. Elsayed T, Lin J, Oard DW (2008) Pairwise Document Similarity in Large Collections with MapReduce. 265–268.

9. Bin L, Yuan G (2012) Improvement of TF-IDF Algorithm Based on Hadoop Framework. Proc 2nd Int Conf Comput Appl Syst Model 391–393. doi: 10.2991/iccasm.2012.98

10. Bank J, Cole B (2008) Calculating the Jaccard Similarity Coefficient with Map Reduce for Entity Pairs in Wikipedia.

11. Wan J, Yu W, Xu X (2009) Design and Implement of Distributed Document Clustering Based on MapReduce. 7:278–280.

12. Zhou P, Lei J, Ye W (2011) Large-Scale Data Sets Clustering Based on MapReduce and Hadoop. 16:5956–5963.

13. Mihalcea R, Corley C, Strapparava C (2005) Corpus-based and Knowledge-based Measures of Text Semantic Similarity.

14. Turney PD (2010) From Frequency to Meaning : Vector Space Models of Semantics. 37:141–188.

15. Raghavan V V., Wong SKM (1986) A critical analysis of vector space model for information retrieval. J Am Soc Inf Sci 37:279–287. doi: 10.1002/asi.4630370502

16. Terms RT NRC Publications Archive Archives des publications du CNRC Coherent Keyphrase Extraction via Web Mining Coherent Keyphrase Extraction via Web Mining *.

17. Kalaivendhan K, Sumathi P (2014) An Efficient Clustering Method To Find Similarity Between The Documents. 2:2532–2535.

18. Shvachko K, Kuang H, Radia S, Chansler R (2010) The Hadoop Distributed File System. 2010 IEEE 26th Symp Mass Storage Syst Technol 1–10. doi: 10.1109/MSST.2010.5496972

19. Lin X, Meng Z, Xu C, Wang M (2012) A Practical Performance Model for Hadoop MapReduce. 2012 IEEE Int Conf Clust Comput Work 231–239. doi: 10.1109/ClusterW.2012.24

20. Ekanayake J, Pallickara S, Fox G (2008) MapReduce for Data Intensive Scientific Analyses. 2008 IEEE Fourth Int Conf eScience 277–284. doi: 10.1109/eScience.2008.59

21. Dean J, Ghemawat S MapReduce : Simplified Data Processing on Large Clusters. 1–13.

22. Lämmel R (2008) Google's MapReduce programming model — Revisited. Sci Comput Program 70:1–30. doi: 10.1016/j.scico.2007.07.001

23. Zaharia M, Chowdhury M, Franklin MJ, et al. Spark : Cluster Computing with Working Sets.