Brute Force and Indexed Approaches to Pairwise Document Similarity Comparisons with MapReduce


Lukas Lewandowski
University of Konstanz
lukas.lewandowski@uni-konstanz.de

ABSTRACT
Matching documents based on similarity is a common and important use case in world wide web search and thus in Information Retrieval. When a user discovers a web page or paper which interests him or her, the user is probably interested in similar documents which contain further information to the previous searched topics. Therefore search frameworks exist, such as PubMed which offer users a top k list, containing related documents to the searched paper. This work introduces a scientific paper presented at SIGIR, 2009.

1. INTRODUCTION AND BASICS
Similarity computation is a common way in Information Retrieval to analyze similarities between two or more documents. Such a computation can simplify search of relevant information. A user can search i.e., a scientific paper with some keywords and afterwards, based on a relevant match, similarity computation offers a possibility to depict the most similar documents. This scenario can be also found in PubMed where a user searches for a paper and gets a top 5 list of similar documents. With this top 5 list it is easier to find relevant information to a given topic. Jimmy Lin introduces in his paper Brute Force and Indexed Approaches to Pairwise Document Similarity Comparisons with MapReduce three algorithms for pairwise similarity comparison in a scalable distributed environment, based on MapReduce. Lin used the test document set PubMed’s MEDLINE genomics track (2005), an authoritative collection of about 4.59 million abstracts. This paper is organized as follows: in the following subsections the author introduces some basics recording to MapReduce and Information Retrieval, which are needed to understand the algorithms in section 2. Section 3 gives an overview about the experimental results and finally section 4 compares Lin’s ideas with other solutions and summarizes the approach ideas.

1.1 MapReduce and Hadoop
When the amount of data is growing fast a similarity computation is not possible on an ordinary personal computer due to the limits of RAM, CPU and hard disk. To cope with such situations a common way is to distribute the data. Google introduced some years ago MapReduce as a scalable distributed environment. MapReduce is running on a distributed file system as Hadoop’s HDFS. Hadoop is an open source implementation of Google’s MapReduce and Google File System. The main idea is to perform a record algorithm on data nodes in parallel (map) and afterwards to aggregate the intermediate results (reduce). The results are then written to the distributed file system. This situation is depicted in Figure 1. Furthermore Hadoop is responsible for load balancing and replication, thus a programmer can concentrate completely on implementing the map and reduce functionalities.

1.2 Inverted Index
An inverted index is a core component in Information Retrieval systems. The main idea is to match a query term to all documents to achieve a term - list of documents containing the term - matching. Therefore term frequency and weight can be calculated on a fixed list of documents and unnecessary computations on documents, which do not contain the term, are avoided. Furthermore when several search terms are available it is only needed to perform an AND join and the list of relevant documents is minimized. Such an example of an inverted index is depicted in Figure 2.

1.3 Weight of Terms
To perform similarity computation an often used approach is to first compute term frequency (tf) for each document

Permission to make digital or hard copies of all or part of this work for personal or classroom use is allowed.

Seminar at DBIS Group: Information Retrieval W2010, February 2011; Konstanz, Germany

2http://hadoop.apache.org/
Figure 2: An inverted index consisting of terms as keys and postings as values.

and document frequency (df), which means the amount of documents in the collection where the term occurs, \[2\]. These factors can e.g., mark a document with higher tf for a given term as more relevant. On the other side df can protect of terms which are not a good similarity factor, because they occur e.g., in almost all documents in the collection. To prevent that terms which have a big df value and a false interpretation of tf have significant influence on the result, a weight score function has been introduced. In (1) the weight \(w\) function for a term \(t\) in a document \(d\) is depicted. \(N\) is the amount of all documents in the collection.

\[
w_{t,d} = (1 + \log tf_{t,d}) \cdot \log \frac{N}{df_t} \quad (1)
\]

2. PAPER APPROACH

This section concentrates on Lin’s MapReduce algorithms to perform pairwise similarity comparison in a distributed environment. The Brute Force approach performs similarity comparison direct on term vectors of each document. Parallel Queries and Posting Cartesian Product create in a preprocessing step an inverted index for their posting lists and use it in their approach to compute the similarity of documents. All computations are implemented in map and reduce functions. The next subsection introduces the similarity function and afterwards the three algorithms for similarity computation are described.

2.1 Pairwise Similarity of Documents

With the introduced function in (1) tf-idf weights for each term \(t\) in each document can be computed. To perform similarity comparison of two documents it is only needed to multiply the weights for a term from a given document with the weight for the same term in another document. The intermediate results (weights) have to be summed up for all terms in the vocabulary. This situation is depicted in (2).

\[
sim(d_i, d_j) = \sum_{t \in V} w_{t, d_i} \cdot w_{t, d_j} \quad (2)
\]

Furthermore it is not needed to go through the whole vocabulary of terms, but only for relevant terms that are common in both as an intersection of terms from document \(d_i\) and \(d_j\). This is because if one weight would be zero, because of no occurrence of a term the multiply will be also zero.

2.2 Brute Force

The Brute Force (BF) approach does not need an inverted index. The map function gets a document as input. This input document \(d\) has to be compared to other documents from the collection. For this purpose a block of documents (document term vectors) is loaded into the data nodes main memory. This is because an ordinary data node cannot load the whole collection into main memory, due to the big collection size. For the similarity comparison the block of document term vectors will be iterated in a loop and each document vector from the block will be compared with the input document \(d\). If the computed score is bigger than zero, the document id from the block will be added as key and the document id \(d\) with the similarity score, which has been computed between these both documents, as value as intermediate result. This situation is depicted in Figure 3.

Figure 4 depicts the reduce function which gets intermediate results from the mapper. It gets a document id and a list of similarity scores to other documents as input. Afterwards it initializes a priority queue to offer a top \(k\) list. For this purpose it iterates simply over the input list and inputs the similarity scores into the top \(k\) list. The result will be written to disk.

2.3 Parallel Queries

As indicated before Parallel Queries and Postings Cartesian Product use an inverted index to perform their similarity computation. The inverted index will be created through a MapReduce job, too. The mapper simply gets as input an document and iterates through its terms and counts the term frequency. Then the term will be output as key and document id - tf - matching as value. The reducer gets as input the term with its list of documents in which the term occurred. The list also contains information about tf. It initializes a posting list and maps the list of documents with tf to it. Afterwards the postings list will be sorted based on tf. The output is then the term as key and the postings list as value.

Parallel Queries (PQ) is simple and tries to treat pairwise similarity comparisons as a very large ad hoc retrieval problem. The algorithm shown in Figure 5. The map function

1: procedure Map(a, d)
2: \([\{b_1, e_1\}, \{b_2, e_2\}, \ldots, \{b_n, e_n\}] \leftarrow \text{LoadDocuments}()\)
3: for all \((b, e) \in [\{b_1, e_1\}, \{b_2, e_2\}, \ldots, \{b_n, e_n\}]\) do
4: \(s \leftarrow \text{ComputeScore}(d, e)\)
5: if \(s > 0\) then
6: \(\text{ Emit}(b, (a, s))\)

Figure 3: Map function of the Brute Force approach.

1: procedure Reduce(k, [(a1, s1), (a2, s2) \ldots])
2: \(\text{ Initialize.Priority.Queue}(Q)\)
3: for all \((a, s) \in [(a1, s1), (a2, s2) \ldots]\) do
4: if \(Q.\text{Size}() < k\) then
5: \(Q.\text{Insert}(a, s)\)
6: else if \(s > Q.\text{Min}()\) then
7: \(Q.\text{ExtractMin}()\)
8: \(Q.\text{Insert}(a, s)\)
9: \(\text{ Emit}(b, Q.\text{ExtractAll}())\)

Figure 4: Reduce function of the Brute Force approach.
gets a term and its associated postings list as input. Then it loads a block of documents in main memory, which are interpreted as queries. Afterwards it iterates over all queries and checks if each query contains the input term. If so, an inner loop iterates over the postings list and computes the partial similarity score for the term of other documents. Then the query id will be written as key and the partial scores according to the term as value (accumulators) as intermediate result.

The reducer, which is depicted in Figure 6, gets all partial similarity scores to a given query as input. It iterates over all partial scores and computes the total similarity to other documents.

2.4 Postings Cartesian Product

The last introduced algorithm is Postings Cartesian Product (PCP). It uses the same postings list created one step before, described in section 2.3. The map function gets as input a term and its postings list, see Figure 7. It calculates pairwise similarity directly from the postings. The main idea is to generate partial scores for a particular term by taking the Cartesian product with itself. It iterates in an outer loop over the postings and computes similarities in an inner loop over the same postings. This approach has to take care to avoid computation of the own document. The document id will be written as key and the partial scores as values as intermediate result.

The reducer works completely identical to the reducer described in section 2.3. It gets the document id as key and the partial scores as values and sums up the partial scores to complete similarity scores. The result will be written to the output. This situation is shown in Figure 6.

3. EXPERIMENTAL RESULTS

In this section the author will present used optimization techniques for the Brute Force approach and approximation techniques for Parallel Queries and Postings Cartesian Product. Both optimizations and approximations will increase the efficiency of the similarity comparison, but approximations lead to a not significant decrease of effectiveness. The second subsection will describe Lin’s results based on a top 5 result list.

3.1 Optimizations and Approximations

As mentioned above only the Brute Force approach has a real optimization, because we have no loss of effectiveness. Approximations, which lead to a decrease of effectivity, can be used for all three algorithms.

An obvious optimization in Brute Force approach is to decrease the number of intermediate results, which are shifted from mapper to reducer. Since Lin is searching for the top 5 related articles in PubMed, it is sufficient to shift only the 5 best scores from a mapper to the reducer. This improvement reduces vast amounts of unnecessary network traffic.

The following approximations can be performed on the introduced algorithms: first, reduces the number of accumulators. In PQ and PCP an accumulator is created for every posting, which results in tremendous amount of data that needs to be shifted between mapper and reducer. PQ and PCP adopt a simple hard limit strategy, which means that query processing stops as soon as a defined limit is reached. Since postings are frequency sorted, the limited accumulators hold documents with largest contributions. The second approximation can be used for BF and PQ, where only top n terms in a document are considered. That means that Lin defines a term limit based on document frequency, i.e., only the best 80 terms with a low df. The last approximation is applicable to PQ and PCP. It is a simple hard limit based on document frequency, where all terms with document frequency greater than a threshold are simply ignored.

3.2 Results

All algorithms use MapReduce as distributed computation model, which is implemented in the open source Apache high level project Hadoop. The actual PubMed term weighting model [4] was used to mirror the motivating application. The experiments were run on a cluster of 480 machines and used Java 1.5 and Hadoop version 0.17.3. To measure approximations, effectiveness experiments were conducted with the test collection from TREC 2005 genomics track, which contain 4.59m records. For the evaluation fifty topics were developed based on interviews with biologists and can be considered as representative. Abstracts relevant to the same topic were assumed to be relevant to the test abstract. For both effectiveness criteria micro- and macro-averaged values were computed.

Brute Force approach performed best, when exact similarity is needed. PQ and PCP are not competitive to BF, when exact similarity comparison is performed, due to memory constraints and vast amount of network load. Therefore Lin only presents BF as algorithm for exact similarity. The results depicted in Figure 8 show that approximations perform much better than BF, when efficiency is considered.
The efficiency gain of the approximations goes with effectivity loss, but the decrease is only between 1-5%.

4. COMPARISON AND CONCLUSION

This section describes some comparisons to other approaches and gives a short conclusion.

4.1 Comparison to other Approaches

Unfortunately Lin does no comparison of his algorithms to other techniques. It would be interesting to see performance evaluation with competitive approaches. Vernica et al. [5] describe also a join based similarity comparison based on self joins and R-S joins. Furthermore the authors introduce techniques to distribute data in the network. Another interesting technique is presented in [6] where similarity search is performed with the use of hashing techniques. The authors describe how hash functions can be taken to group documents together and that documents with similar hash values can have similar contents. Furthermore it would be interesting to know if relevance feedback, like introduced in [7] can increase Lin’s approaches due to effectivity or efficiency. Another idea by Zhang et al. [9] is to search for partial duplicate detection based on sequence matching.

4.2 Conclusion

Lin introduced pairwise similarity computation based on MapReduce, a distributed computation model. The author described three different algorithms to cope with similarity comparison in distributed environments. Brute Force’s exact similarity performs best, when effectivity of results is important. It uses no inverted index to perform computation. On the other side Parallel Queries and Postings Cartesian Product use an inverted index as input to compute similarity values. These both algorithms performs best according to efficiency, because the effectivity has been limited by some approximations. MapReduce is a distributed computation model which is used more often in Information Retrieval technique evaluation, where a distributed environment is important, because the scientist can concentrate on implementing his / her approach within the map and reduce operations. An interesting evaluation on that is given in [8], where the authors compare this base for programming with existing Information Retrieval systems like Lemur, Lucene or Terrier.