Journal of Computers Vol. 28, No. 3, 2017, pp. 251-264
doi:10.3966/199115592017062803020

Efficient Parallel Method for Documents Similarity in a Large Dataset

Niyigena Papias[1], and Zuping Zhang[1*]

[1] School of Information Science and Engineering, Central South University, Changsha, 410083, PR China
papiasni@yahoo.fr, zpzhang@csu.edu.cn

Received 12 October 2016; Revised 06 March 2017; Accepted 30 March 2017

Abstract. Near-duplicate document detection attracts much attention from researchers since the growth of documents production is very high. The main problem confronted while looking for duplicate or near-duplicate document detection is a very high dimensional data which increases the time and space requirements for processing the data. With the trend of production of new documents, the system to detect similarity among documents becomes almost impracticable. We are proposing a new approach for solving this problem which consists in reducing the dimensionality of data and also use efficiently parallel programming to fully maximize the available capacity of the hardware. The intuition we have by using parallel programming is that more processors/core will perform better than only one processor if their management is well done. We have implemented our method and tested it empirically and experimental results have demonstrated that our algorithm performs better than other methods used for All Pairs Similarity Search (APSS) which employ multi-core and multi-programming to deduct the similarity of the documents. The results show that our method can reduce up to 65% terms to be used in similarity computation and its execution time is better than Partition-based Similarity Search method which uses parallel processing for document similarity.

Keywords: dimensionality reduction, document similarity algorithm, pairwise document similarity, parallel programming, query likelihood

1 Introduction

Near-duplicate document detection has attracted the attention of many researchers for a few decades ago. Nowadays, the development of technology for processing and storing data has allowed the production of billions of documents and web pages stored in the cloud. This way, the system to determine near-duplicate document has a big number of terms to be computed. Applications for Near-Duplicate Document Detection use many data to identify objects or set of objects similar one to another under a specified threshold. Given the massive data to be involved, to determine document similarity becomes a time-consuming process [1, 10]. Recently, researchers have proposed different methods to solve this problem but as the number of documents increases the performance of these methods suffer. For example, the method which uses shingles even if has a very good precision, it becomes impractical for a large corpus. TF-IDF also has been very efficient as metric to calculate the similarity between documents but fails for big datasets because the increase in the number of documents implies an exponential increase in the number of terms to be considered in the computation. The exact method which compares every term in two different documents is also impractical and fails for few documents only. In this paper, we are proposing a new method which considers the reduction of the terms to be used in the corpus so that the comparison of terms is applied to a subset of terms representing the large dataset.

The objective of Pairwise Document Similarity detection is to determine which document is related to another one in the corpus, having the threshold to be used for making such decision. This task becomes very complex when it is performed on large corpus because each document is compared to every document in the corpus, giving a Cartesian product with execution time of Ω (2^n). The function of document similarity gives the importance on a word in the document which determines if the word should be used in similarity computation or not [11, 18]. This is very important for our method because

* Corresponding Author

two documents may not be similar by comparing every word but still have the same content and only a small portion differentiates them or some parts of the document are relocated to different places within another document. The objective of our method is to identify that important content within the document and efficiently use it to determine the similarity among the documents. Because the document similarity detection system needs to handle a very large collection of documents, an ideal algorithm for such system should give a response in real-time. Thus, efficiency has been the main focus on different approaches solving the similarity problem. Different methods have been implemented by researchers to solve this problem, but sophisticated data structures and clever algorithms optimization alone are not sufficient anymore.

In this paper, we are presenting a new method to approach this complex problem by performing efficiently the similarity computation of documents. The contribution of our method can be summarized as follows: the first is to reduce the dimensionality of the large collection of documents to a subset of terms. This subset is selected to represent the large document collection with the intuition that applying similarity function to reduced terms will increase the efficiency of the process; the second contribution is to exploit the available processing power of parallel computer using Multi-core, GPU, and cluster of computer, and the third contribution is to use best data structures to allow optimized algorithm and to let it be shared among several processes executed by multiprocessor of the machine and avoid inter-processor communication. The results of our proposed method have shown that our method performs better than previously proposed method, even better results for large data collection which justify the scalability of our method.

2 Related Work

In the past few decades ago, many types of research have been done for document similarity. The methods in the state of the art can be grouped into two main categories: heuristics methods and exacts methods. In exact methods, researchers try to find an exact solution as in the context of a database but the practicability is minimal. For Heuristics methods, researchers use different techniques to give a solution to this problem as hashing, shingling, and dimensionality reduction. In this category of research, Charikar has defined in [15, 17] a hashing scheme as a distribution on a family of hash functions operating on a collection of vectors in which for any two vectors, the probability that their hash value becomes equal is proportional to their similarity.

The large data set available and computation nature of pairwise similarity are a limiting factor for the Near-Duplicate detection applicability to large-scale real-world problems and calls for alternative approaches. Given a corpus of documents in a d dimensional space, existing pairwise similarity search algorithm compute $O(n^2)$ similarity scores [2, 16]. With the trend of data production and data availability, a good approach should not be limited to the optimization of the algorithms used but also should consider the new technologies offered by the hardware and use both techniques for an efficient solution.

The hardware performance improvements are currently achieved not by faster processors but by increasing the number of processor cores. This yields adaptive algorithms which take this advantage and use the parallelism in order to solve problems which were before resources greedy. Nowadays, historic difficulties of parallel programming are almost overcome: the cost of parallel systems has significantly decreased and, in fact, by 2008 it was becoming difficult to find a single CPU desktop system, with a single core, CPUs being relegated to notebooks and embedded devices. By 2012, even smartphones were starting to support multiple CPUs [13].

One goal of parallel programming among others is performance. Traditionally, the main goal of parallelization was to speed up the computation needed to solve a particular problem by engaging several processors and dividing the total amount of work among them [3]. Given the recent trends on the part of all major manufacturers towards multicore/multithreaded systems, parallelism is the way to go for those wanting the avail themselves of the full performance of their systems. For decades, different types of researches have been done to use in a perfect way parallelism for effective heuristic optimal or near-optimal solutions to difficult optimizations problems. Algorithms have been designed for exploiting high-performance and parallel computing resources for randomized iterative evolutionary computation [3, 9].

High performance and parallel computing have been extensively studied to tackle the previously mentioned computational challenges in Near-Duplicate document detection because this application has a very good application field for parallelism. With exponentially growing data sets and modern

multiprocessors/multi-core system architecture, it is obvious that serial solutions for document similarity detection are the major rate-limiting factor for the applicability of similarity detection for large data collections, so parallelization becomes inevitable. In this context, Awekar et al. [2] were interested in the implementation of a parallel solution of All Pairs Similarity Search for shared memory multi-processor machines. In their method, the inverted index to use in similarity computation is shared among processors. They have provided different techniques to optimize the inverted index code performance and ease Map Reduce paradigm. However, even with extensive optimization to filter unnecessary computation, MapReduce communication with the inverted index can increase quadratically as the data set size scales up and can incur significant synchronization and I/O overhead. To improve the method presented above, Kulkarni [9] has extended an alternative selective search, an approach that partitions the data set based on document similarity to obtain topic-based shards with few computational resources. In his approach, only a few shards that are estimated to contain relevant documents are searched. Even though this approach achieves a good level of effectiveness, its efficiency suffers for large data set. Our method brings an improvement to come up with a solution to these weaknesses: the corpus is reduced, thus, the superset of terms in documents collection is represented by a subset of terms which have the same power to represent the documents. With existing methods even though the index is shared in multiprocessor, the locking mechanism between different processor is hindering the performance of the method. Our method allows all the processes to access simultaneously the data structure created to eliminate the idle time of the processors due to the waiting time needed to avoid data race.

We show how to exploit massively parallel high-end computing resources for solving document similarity problem which belongs to a classic combinatorial optimization problem. To achieve this objective, our method uses index sharing technique and divides the computation into independent tasks over the central concurrent dictionary shared across all processors as a read-only data structure. The approach performs an early removal of unwanted comparisons which eliminate and reduce a significant amount of I/O, memory access, and computation. To compute the similarity among documents, our method allows multiple processors to access the created data structure in parallel and real-time. It is clear that the design of the new data structure will not need locking scheme which allows concurrent access to move "freely". The off-line creation of the data structure will use lazy values to avoid the long wait of tasks while locking to avoid concurrent update of the same variable by multiple processes at the same time.

3 Efficient Parallel Document Similarity Detection Method (EPDS)

3.1 The Representation of Document for Similarity

Document similarity system performance depends on the choice made while representing a document; thus a more comprehensive representation of a document leads to a more efficient system. The design of the method we are proposing will use a better document representation to ease document similarity computation. The state of the art has defined several methods to represent a document in Information Retrieval. Among them, the approach using probability ranking principle has achieved a considerable recognition in the last few decades, because the probability is the best method to be used when representing and implementing the uncertainty.

The simple way to detect near duplicates is to sequentially submit each document from a collection as a query to the system to search for highly similar documents in the same collection. By the use of an inverted index as adopted in most Near-Duplicate detection algorithms, the time complexity for this process is sub-linear (to the number of documents in the collection), however, it can be $O(n^2)$ in the worst case. The main objective of our method is to rank documents optimally given a document so that similar documents would be ranked above no similar ones. To allow the system to have the best efficiency, our method uses Language Model which refers to a probabilistic model in the text and defines the probability distribution of terms in a given document. In this modeling, as proposed by Ponte and Croft, the query is assumed to be a sample of words drawn according to a language model estimated based on a document [6]. Thus, two documents will be considered as similar if their Language Model gives the highest probability.

To get the result, we need to define both language model for the document considered as a query and other documents in the collection to be able to sort out the probability. Let consider Q as a query, D as a

document, and θD as a language model estimated on document D. The score of D is a conditional probability $p(Q|\theta D)$, in other words it means Score(Q, D) = $p(Q|\theta D)$. According to this score, we define a binary random variable $X_i \in \{0, 1\}$ for each word w_i to indicate whether word w_i is present ($X_i = 1$) or absent ($X_i = 0$) in the query. Thus model θD would have precisely $|V|$ parameters, i.e., $\theta D = p\ (X_i = 1|D)\ i \in [1, |V|]$, which can model presence and absence of all the words in the query. Thus, according to this model, the query likelihood can be written as

$$P(Q \mid \theta D) = \prod_{w_i \in Q} p(X_i = 1 \mid D)\,. \tag{1}$$

In this formula, the first product represents the words in the query and the second words not occurring in the query. To capture TF, we can treat each word w_i in D as a sample from our model where only w_i has shown up and all other words are absent. Thus according to the maximum likelihood (ML) estimator, $p(X_i = 1|D)$ is equal to the relative frequency of word w_i in D, i.e.,

$$p(X_i = 1 \mid D) = \frac{c(W_i, D)}{|D|} \tag{2}$$

Where **c**(w_i, D) is the count of word w_i in D and $|D|$ is the length of D (i.e., the total word)

You may prepare your camera-ready manuscript with MS-Word using this typeset together with the template joc.dot (see Sect. 3) or any other text processing system. In the latter case, please follow these instructions closely in order to make the volume look as uniform as possible.

Our method uses the probability model to efficiently sort out similar documents in a large data set. Like in other All Pairs Similarity Search process, our method considers every document in the collection to be represented as a query and is compared to the rest of documents in the same corpus. Thus, our method uses "Language Model" to obtain the weight of every term in every document and then rank the documents according to the weight of their respective terms. Only documents with the probability of similarity greater or equal to the threshold will be maintained. To rank the documents, we determine the probability by which the query (represented by a document) can be generated by another document language model [8]. As we want to produce the documents responding to the query, we would in general like to calculate $P(D|Q)$ to rank the documents. Using Bayes' rule, the probability is given by the formula:

$$P(D \mid Q) = P(D \mid Q)\ P(D). \tag{3}$$

The right-hand side is rank equivalent to the left-hand side (i.e. we can ignore the normalizing constant $P(Q)$. $P(D)$ is the prior probability of a document and $P(Q|D)$ is the query likelihood given the document. The probability of ranking a document is $P(Q|D)$ which is given by the following formula

$$P(Q \mid D) = \prod_{i=1}^{n} P(q_i \mid D) \tag{4}$$

Where q_i is a query term, and we know that there are n words (or terms) in the query. To compute this score, we need to have estimates for the language model probabilities. The obvious estimate would be

$$P(q_i \mid D) = \frac{fq_i, D}{|d|} \tag{5}$$

Where fq_i, D is the number of times word f_i occurs in document D, and $|D|$ is the number of words in D. For a multinomial distribution, this is the maximum likelihood estimate, which means this is the estimate that makes observed value of fq_i, D most likely.

The execution time of exact method to determine Pairwise document similarity is quadratic to the data size and thus does not scale well for a big dataset. Our method reduces the execution time by using two techniques: data size reduction and parallel processing. While processing the similarity, all the terms in the corpus will not be considered because it's obvious that some terms in the documents cannot help for determining its similarity to other documents. These terms are considered as noise, removing them from the text cannot affect the determination of document similarity but will reduce considerably the size of text to go through in the computation. Moreover, our method presents an efficient algorithm which considers the input of hardware new technologies: with the recent trends on the part of all major manufacturers towards multi-core/multithreaded systems, parallelism should be considered if we want to

fully benefit from the performance of the systems.

Fig. 1 shows the overview of the proposed method. It shows that the main input of parallel processing in document similarity is the creation of the index to be used during the process. Many processors can work in parallel to build the index without interfering each other. In serial processing, the creation of such index takes more time because the process considers every term in the document which increases the execution time. Again our method use parallel processing to process the query: the index created have an array of all the terms in a document collection and every term has a posting list associated with the term. Every list contains all documents in which the term appear and the weight of the term in the document. We have seen before that every document is considered by our method as a query, thus, to compute the similarity between a query with a document, the system fetch all posting lists for every term in the query to compute the probability according to the weight of every term. Because the query document has many terms, our method uses parallel processing to retrieve and compute the probability because there will not be any interference between processors.

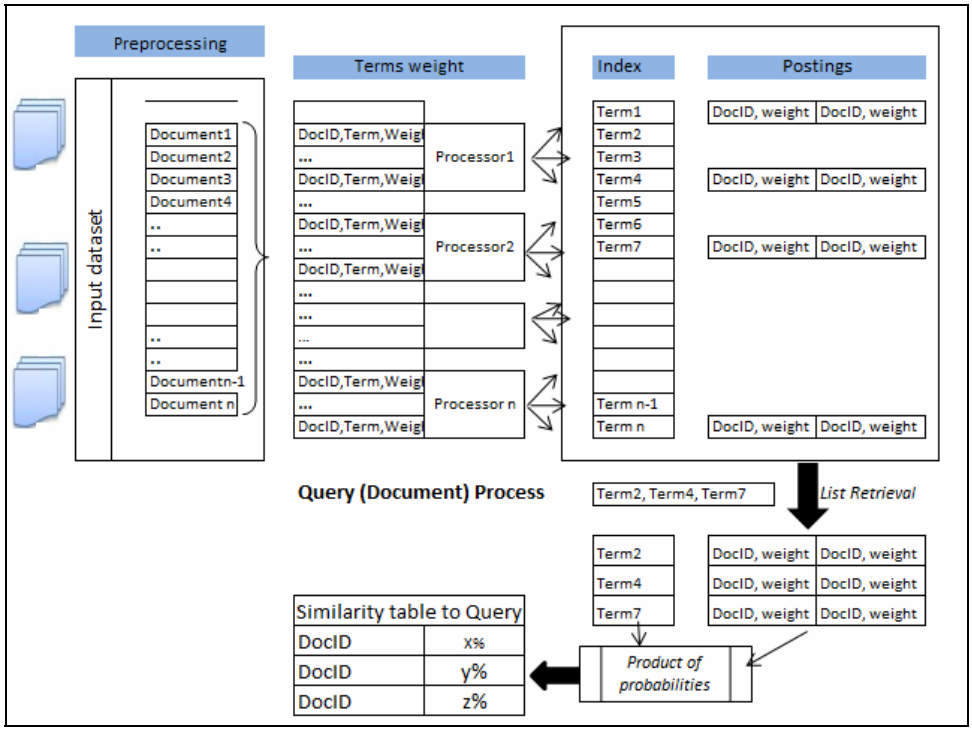

Fig. 1. EPDS method overview

3.2 Reducing Dataset Size

Document dimensionality is the first problem that hinders document similarity systems. The system is dealing with millions of documents and thousands of billions of terms. To give an efficient solution, our method reduces this dimensionality by representing every document in the collection by a set of important phrases to be considered in the place of the document. By taking n important phrases in the document, we expect to capture the whole idea in the document with an intuition that if two documents are having the same important phrase it means that they are similar. Our method reads all documents in the corpus, but only top n phrase in every document are kept. The algorithm to perform this task uses a word frequency in the given document. After preprocessing of the text (removal of all noise in the document, i.e. stop words, punctuation and words containing only a few characters), the frequency of each word is computed and kept in the dictionary. This data structure is chosen because it is fast and allows a constant access time $\Theta(1)$ which will help our method to speed up the access of term. To determine which sentence to be considered in this process, our method uses the weight of every term in the document, so the sentence which has terms with high weight will be chosen to represent the document. The following algorithm implements this idea.

Create_Documents Summary Algorithm

```
//This algorithm summarizes documents in corpus by representing each
document by its top n phrases
Input: The Corpus to summarize
       Paral_Level -> The number of processors/cores to be used
Output: Dictionary of Document summary

LPhrase = Empty list
LFrequencies = Empty list
Reader : The corpus reader
ReducedCorpus: The empty Corpus
//Get the list of all documents names from corpora
For i From 1 to Paral_Level
      //Obtain the small number of document in corpus to be summarized
by a task
      Srart_Task (
      ReducedCorpus = Get_nDocuments in Corpus(Paral_Level)
        For each Doc in ReducedCorpus
            Scores = []
               Doc = RemoveStopWords(Doc)
               Doc = Steemer(Doc)
      ListSent = SentenceTokenizer(Doc)
      For each word in Doc
            freq = Get the word frequency in the document
            DictSum.Add (word, freq)
      End for
      For each word in ListSent, i: Counter
            If word in DictSum then
                  Scores[i] = Scores[i] + DictSum[word]
            End if
      End for
        DocSummary = Get top sentences in Scores
        ReducedCorpus.Append(DocID, DocSummary)
      End for
    )
End For
//Create the language Model of the LFrequencies (every term is given
its weight)
For i From 1 to Paral_Level
      Srart_Task (
              LWords = GetWordTokens (ReducedCorpus)
      For each w in LWords
          Freq = Find_Frequency ( w )
          LFrequencies.Add (w, Freq)
      End for
      )
End For

//The dictionary can be written to the file for the future use
Return (Dict)
```

After the summarization of the documents in the collection, our method proceeds by creating the Language Model of the reduced corpus which will be used in Near Document Similarity Detection. As the document is represented by top n sentences, so the term should be normalized to the total number of documents which remains after the reduction of documents to n sentences representing every document.

This weighting scheme used by our method is also known as *TF-IDF* in Information Retrieval. In the previous section, we have seen that to determine the similarity between documents, we should compare the Language Model of both documents; so to allow the system to perform this comparison, there are important data we need to keep in the concurrent dictionary such as term, documentID, and the weight (representing the probability of the term).

The creation of this data structure will be in parallel to reduce the execution time used by the serial method. In this scheme, the number of documents to be considered in the collection is divided into different groups which will be allocated to each processor/core. These cores are using a concurrent dictionary which allows them to access the dictionary in shared memory without the data race problem and any other communication between the processors. This is very important for execution time reduction because there will not be a waste time by any processor in waiting for the locking procedure needed in memory shared data structures or the management of extra communication among different process. The algorithm to implement the idea is the following.

Create_ProbabilityDistribution Algorithm

```
//This algorithm create a dictionary for terms in documents given their probability distribution
Input: LFrequencies The Probability Distribution list for all documents in corpus
     Paral_Level -> The level of parallelism
     Output: Dictionary, data structure to help parallel search to compute documents similarity
     Dict = Empty Dictionary
     DocSimilList = Empty list

     //split the task in parallel tasks
     For i From 1 to Paral_Level
          Srart_Task (
          For each PDistr in LFrequencies
              PostingTerm = [Array(PDistr.Document, PDistr.Weight)]
              Dict[PDistr.Term].Append (PostingTerm)
          End For
          )
     End For
     Return (Dict)
```

3.3 Pairwise Computation

The creation of dictionary representing the Language Model is an off-line process and should be timely updated to take into consideration new documents in the collection. The process to determine the similarity among documents has the Language Model as input to be used to rank the documents according to the query. As we need the pairwise similarity, each document in the corpus is considered to be a query, so the process will need the list of documents ID as well. Documents are considered to be similar if they are sharing a certain number of terms not less than a threshold, so the algorithm needs to create a data structure which will allow linking all documents sharing a particular term. Our method uses a concurrent dictionary to implement this data structure whereby the key of the dictionary is the term of the document while the value is the list of other documents in corpus sharing any term with the document represented as key. This list (value of the dictionary) contains pairs of document ID and the total weight of the shared terms. At this stage, we have the Language Model implemented in a dictionary which will improve access time to any value in the LM since access a value in a dictionary is constant $\Theta(1)$. The algorithm takes the list of all the documents and, in parallel, every Language Model for all documents (considered as a query) is compared to the Language Model of the collection in the dictionary we have created.

The similarity is given by applying the formula (4). The main input of this algorithm is the computation of similarity in parallel. Actually, the formula is not directly applied by one process but

many processes will contribute to the computation of the formula: a concurrent dictionary is used to be updated by several processes each. The list of documents in the collection is divided into several chunks which will be assigned to a different process. For a particular process, it will scan every term in the LM of the documents, get the list of documents sharing the same term in the dictionary, compute the similarity probability by using dot product, then add or update the similarity of the two documents in the dictionary. The algorithm concludes the process by eliminating all documents in the data structure with insufficient terms compared to the threshold. At this point, the similarity between documents can be read in the dictionary in a constant time.

Create_Similarity Dictionary Algorithm

```
//This algorithm create a dictionary for similarity. Posting key is
document name, value is the similarity //with the query
Input: DictionaryProb The Dictionary for Probability Distribution
      Query: The query document/Text
      Paral_Level -> The level of parallelism
      Output: List of similarity between query and different documents
sorted in decreasing order.
      //split the task in parallel tasks
      For i From 1 to Paral_Level
            Srart_Task (

            For each Term in Query IN Parallel
                Docs = getDocumetFromPosting(Term)
                For each d in Docs
                      Dictionary[d.filename].Append(d.Prob * q.Prob)
                End For
           End For
           )
      End For
      //Get the list of sorted probabilities
      SortedDoc = []
      For term, Weight in DictionaryProb. Iteritems()
          sortedDoc.Append(term, Weight)
      End For
      SortedDoc = SortedDoc.Sort(Weight).Decreasing
      Return (SortedDoc)
```

3.4 Time and Space complexity

The pairwise document similarity detection is known to be in the family of problems that involves combinatorial objects because it has to generate all subsets of documents collection, comparing each document to the rest. We recall that number of subsets of n element set is 2^n, so the execution time for such algorithms becomes $\Omega(2^n)$. This execution time becomes impractical and need other alternatives to give a solution to the problem. In our method, we are using an approximation algorithm which is not exponential running time even though it is not an exact method. Actually, execution time reduction is one of the most challenging goals of parallel programming. Theoretically, adding extra processors to a processing system leads to a smaller execution time of a program compared with its execution time using a fewer processors system or a single machine[12, 20]. Here we want to highlight how our method reduces execution time for the process of pairwise document similarity. Let's assume that there are m terms in n documents. Starting by the first algorithm, the cost is linear time $O(m)$ for reading the corpus. Since it takes a constant time to find, insert or update, the total execution time to create the list of vocabulary and its weight is $O(m)$. The algorithm continues in creating the dictionary in parallel. For sequential algorithm, the execution time should be $O(m)$ but here, the algorithm use parallelism which reduces considerably the time, so the parallel execution time becomes $O(m/p)$ p being the number of processors to include in parallelism. Speed up is shown in our experiments by comparing the serial

execution time and the parallel execution time. We show how much time a parallel program works faster than its serial version used to solve the same problem. Many conflicting parameters such as parallel overhead, hardware architecture, programming paradigm, programming style may negatively affect the execution [12]. Our method deal successfully with these parameters and the results show that our method performs very well.

4 Experiments

In this section, we are presenting experiments performed to show the efficiency of our method. The aim of our method is to reduce the pairwise document similarity execution time by using all cores present in the system with the intuition that many processors sharing a task will perform better than one processor. For the experiment, we have used a multi-core hardware platform, Intel processor with 4 cores (8 hardware threads) and 4GB memory. As discussed before, our method uses a concurrent dictionary data structure, which uses lazy values to limit the data race among the processes and avoid the use of locking scheme for those processes to work independently. The use of this data structure has a positive impact on the speed of our method as explained in the following section.

The data set used was downloaded from RCV1 which is the public benchmark document corpus for Text Retrieval Conferences (TREC). From this corpus, we have gathered thousands of documents with different size. All stop-words were removed and all words in the data set were stemmed. In this set of experiments, we focus on determining the similarity of a different document each other in the collection. Determine the similarity between documents involve the determination of important portion of every document and decide if two or more documents share the same portion. Distinguishing the important portion of the document becomes a key to improving the efficiency of the task. As discussed before, the main problem for document similarity systems is the scale of data to be used in the computation. To solve this problem our method our method reduces the large collection considering only a given number of phrases to be considered in every document.

4.1 Results Analysis and Comparison

We are comparing our method to other methods within the same family of All Pairs Similarity Search (APSS). In this family, we find the system like collaborative filtering, query suggestion, detecting spams, plagiarism detection, web pages mirrors, near duplicate detection and many others. Among these methods, we have chosen Partition-based Similarity Search (PSS) which uses parallelism to detect all pairs similarity. This method has two steps algorithms and runs on a cluster machine having multiple CPU each. We have implemented the algorithm to run on a single machine with several CPU cores (like our configuration seen above) so that we may perform a fair comparison with our method.

The execution time of pairwise document similarity system depends on the size of the corpora to be considered. As the data set size increases, the execution time also increases. That's why our method reduces the quantity of phrases for every document before it starts the comparison. Each document is represented by a given number of sentences considered as the important portion of the document. It is clear that the choice of this number will affect the computation time, precision and recall. A small number of sentences to represent documents will reduce execution time and will increase the precision at the expense of recall whereas a big number of sentences will use more time to complete the computation and will increase the recall but will reduce the precision. This is the normal trade-off between precision and recall. As shown in Fig. 2, our method present good results on the different number of sentences chosen. The x coordinates show the number of sentences, y shows the time used for similarity search among one thousand documents. We are comparing parallel method (our proposed method), PSS, and the sequential method. We can find that our method performs far better than a sequential method for any number of sentences chosen, the execution time for a different number of sentences is almost linear as shown in the Fig. 2. On the other hand, we have tested the input of parallel computation of our method. We found that the execution time depends on the level of parallelization and the number of core processors involved. Fig. 3 shows the results of the comparison of three methods with a different number of retained sentences to represent a document. In this experiment, we are using a collection of 400 documents. We can observe that our method EPDS has a low execution time which increases slowly as the number of sentences increases.

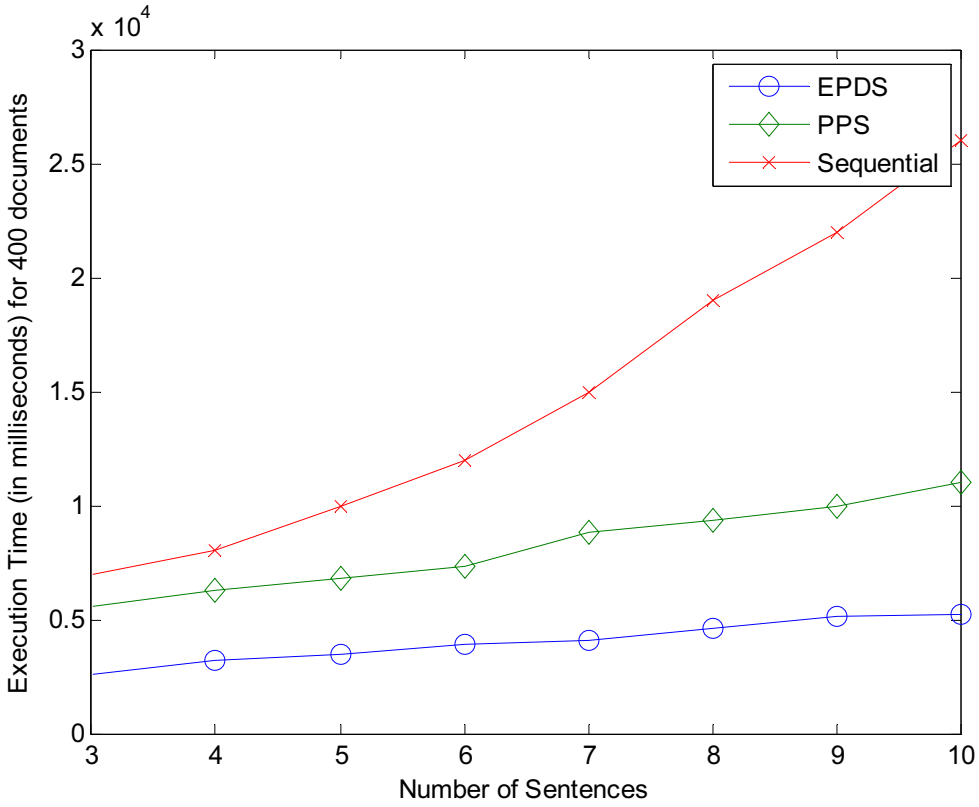

Fig. 2. Execution time comparisons

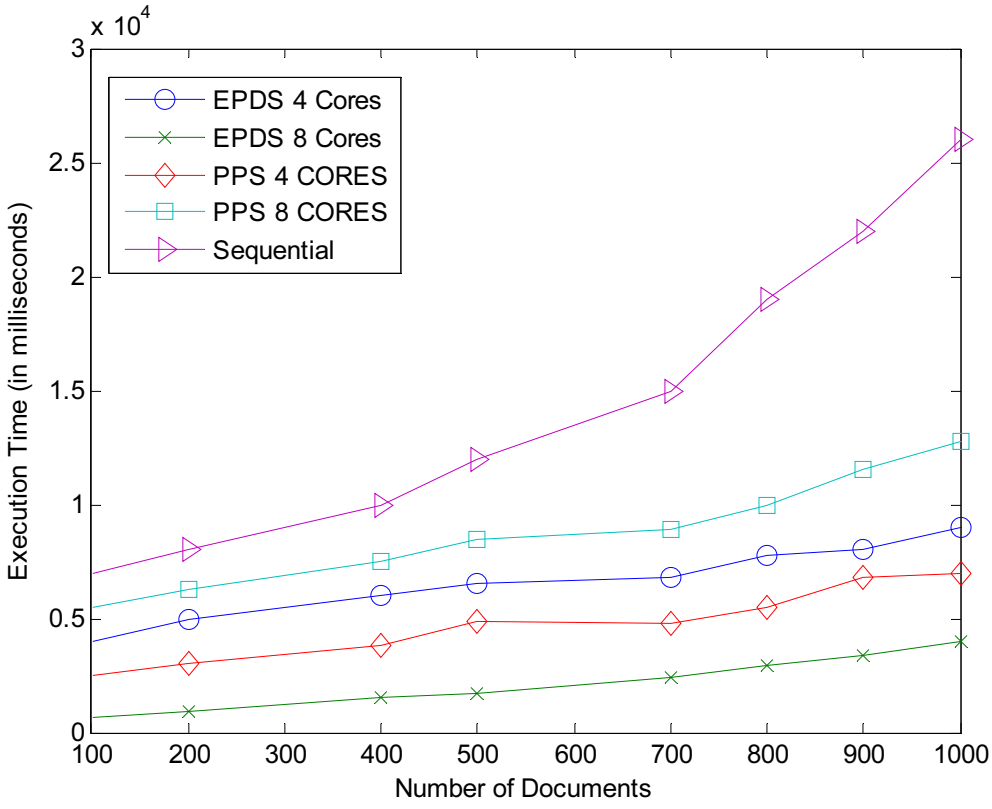

Fig. 3. Comparison of number of processors and sequential

In the following experiment, we present results that test the efficiency of the proposed method for the different size of data set and a different number of processors cores. Again we are comparing the parallel processing EPDS and PSS with the sequential method to perform all-pairs similarity. Fig. 3 shows experiments performed for different documents and different processors cores. The x coordinate shows the number of documents while y coordinate displays the time used in milliseconds. When we compare parallel methods with a sequential method, we can see that two points are to be highlighted: using multiprocessing is better than sequential process whatever number of processors involved in processing [19]. Another thing is that when the number of processors core is increased, the performance increases too. Fig. 3 again shows the results of our proposed method EPDS with PSS which use a parallel mechanism to sort out the similarity. In the experiment, we have limited the cores to intervene in the computation for three methods, first, we limited to four cores and then to eight. Each method was tested to see how it reduces the execution time and for all documents, we have chosen neigh sentences to represent every document in the collection. The Fig. 3 shows that our method reduces more time compared to another method when using four processors/cores and eight processors/cores.

In the following experiment, we present results that show the comparison of an execution time of the proposed method and the existing method. As before, we are comparing our method to two other methods namely PSS and Sequential method using tf-idf because they have proven to produce reliable results in documents similarity in big data sets. Fig. 4 shows, in brief, the results of performance of these three methods. We have tested these methods on different groups of documents ranging from 100 to 1000. Our methods scale very well if compared to other methods because it doesn't consider all sentences in the document; this explains the reason why its trend is almost linear.

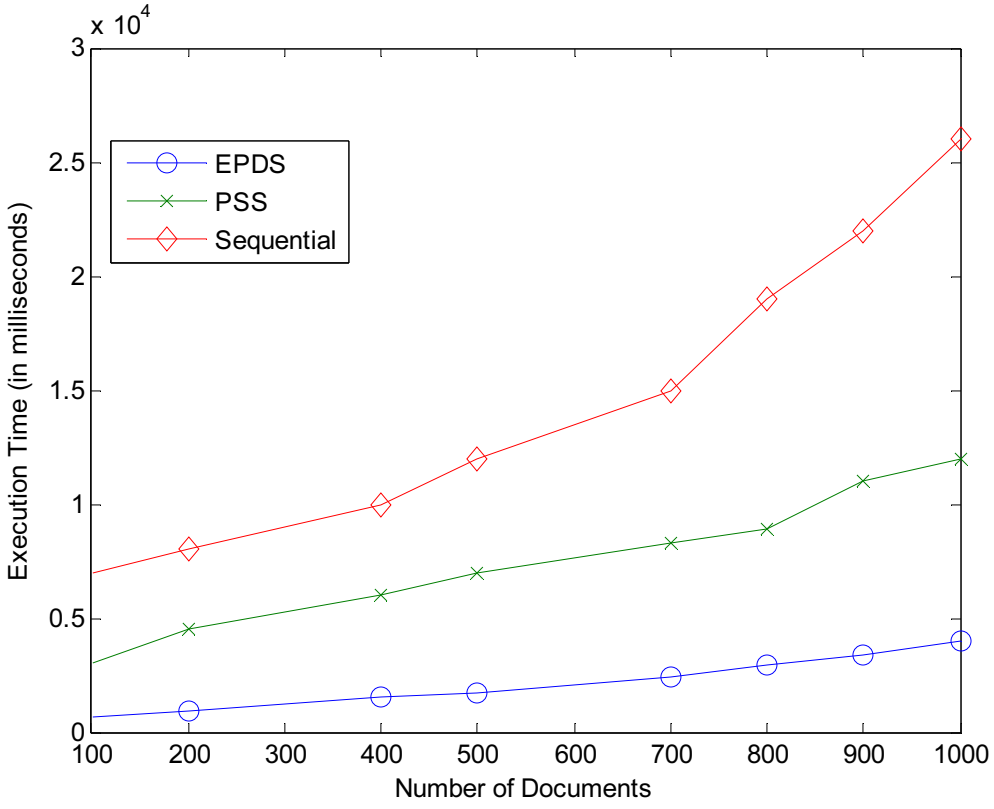

Fig. 4. Execution time comparisons with all pairs method

4.2 Effectiveness

We are using Precision, Recall, and F-Measure to measure the effectiveness of our method. Precision is the ratio of a number of relevant documents retrieved to the total number of irrelevant and relevant documents [21]. The recall is the ratio of the relevant document retrieved to the total number of the relevant document in the corpus. F-Measure is an effectiveness measure based on recall and precision. It has the advantage of summarizing effectiveness in a single number.

Let A be the number of relevant documents retrieved, B be the number of relevant documents not retrieved and C be the number of irrelevant documents retrieved. The Precision (PR), Recall (RC) and F-Measure (F-Measure) are given by the formula:

$$PR = \frac{A}{A + C} \tag{6}$$

$$RC = \frac{A}{A + B} \tag{7}$$

$$F - Measure = \frac{2 \times RC \times PR}{RC + PR} \tag{8}$$

Our method was tested according to a set of 1000 documents in which 6 pairs of documents were near similar. We have performed the test by taking a different number of sentences to evaluate the F-Measure given by our method. As we can see in Table 1, our method has a good F-Measure if we use seven sentences to summarize the documents in the collection. We can see again that fewer sentences affect the effectiveness of our method because the precision is very low for fewer sentences, same situation for more sentences because they affect the recall. Nevertheless, our method performs well when compared to PPS and sequential method as mentioned before.

Table 1. Effectiveness evaluation by Precision, Recall and F-Measure

Number of Sentences (1000 Docs)	Execution Time	Precision %	Recall %	F-Measure %
3	903	29.5	92.3	44.71
4	820	34.6	80.5	48.40
5	870	46.4	73.2	56.80
6	883	52.3	65.4	58.12
7	881	67.8	60.2	63.77
8	922	78.2	51.6	62.17
9	950	83.2	42.8	56.52
10	920	89.2	35.1	50.38

5 Conclusion

In this paper, we presented efficient Near-Duplicate Document similarity algorithms by exploiting the processor cores to compute the similarity between documents in parallel. The method presented in this paper uses probabilistic Language Modeling to determine the similarity between two documents. As the corpus to be considered is very large, our method has represented all documents in the corpus by a given number of most important sentences to reduce the dimensionality of data to be considered. This process is offline so it will not affect the execution time of our presented algorithm as it is considered as preprocessing of the corpus. Considering the hardware performance improvements, our algorithm takes the advantage of multi-core and computes the similarity between documents in parallel. The data structure enabling concurrency and avoiding data race was used to achieve the reduction of execution time up to 65% if we compare our method to a sequential counterpart.

The experimental results validate that the new method achieves significantly higher performance for Near-Document Similarity detection. Since hardware performance improvements are currently and in the near future achieved by increasing the number of processor cores, our new method has the potential to perform even better as the results show that by increasing processor core, the performance increases by an average of 15%. The method presented in this paper is limited to a system with several cores. In the future, we would like to benchmark the performance of our algorithm when applied to other parallel platforms like CUDA and Hadoop which use a cluster of different machines using several processors. If it can be efficiently executed on a cluster we may increase the number of sentences to consider for representing a document and then increase the precision and recall of our method.

Acknowledgements

Project supported by the National Natural Science Foundation of China (Grant No. 61379109, M1321007) and Research Fund for the Doctoral Program of Higher Education of China (Grant No. 20120162110077).

References

[1] M.A. Alabduljalil, X. Tang, T. Yang, Optimizing Parallel Algorithm for All Pairs Similarity Search, in: Proc. International Conference on Tools with Artificial Intelligence, 2008.

[2] A. Awekar, N.F. Samatova, Fast matching for all pairs similarity search, in: Proc. 2009 IEEE/WIC/ACM International Conference on Web Intelligence, 2009.

[3] T. Davidovic, T.G. Cainic, Parallel local search to schedule communicating tasks on identical processors. <https://www.cirrelt.ca/DocumentsTravail/CIRRELT-2013-54.pdf>, 2015.

[4] F. Dehne, H. Zaboli, Parallel real-time OLAP on multi-core processors, International Journal of Data Warehousing and Mining 11(1)(2015) 23-44.

[5] H. Hajishirzi, W.-T. Yih, A. Kolcz. Adaptive near-duplicate detection via similarity learning, in: Proc. 33rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 2010.

[6] J.M. Ponte, W.B. Croft, A language modeling approach to information retrieval, in: Proc. the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 1998.

[7] D. Cai, C. Zhang, X. He, Unsepervised feature selection for multi-cluster data, in: Proc. the ACM KDD'10, 2010.

[8] C.X. Zhai, Statistical language models for information retrieval a critical review, Information Retrieval 2(3)(2008) 137-213.

[9] A. Kulkarni, J. Callan, Selective search: efficient and effective search of large textual collections, ACM Transactions on Information Systems 33(4)(2015) 1-33.

[10] E. Peise, D. Fabrega-Travel, P. Bientinesi, High performance solutions for big-data GWAS. <https://arxiv.org/abs/1403.6426>, 2014.

[11] X. Yuan, J. Long, H. Zhang, Z. Zhang, W. Gui, Optimazing a near-duplicate document detection system with SIMD technologies, Journal of Computational Information Systems, Binary Information Press 3846(2011) 448-451.

[12] A.I. El-Nashar, To parallelize or not to parallelize, speed up issue, in: Proc. IJDPS, 2011.

[13] P.E. McKennye, Is Parallel Programming Hard, And, If So, What Can You Do About It? IBM, Beaverton, 2015

[14] W.B. Croft, D. Metzler, T. Strohman. Search Engines, Information Retrieval in Practice, Pearson Education, New York 2009.

[15] M.S. Charikar, Similarity estimation techniques from rounding algorithms, in: Proc. the Thirty-fourth Annual ACM Symposium on Theory of Computing, 2002.

[16] X. Evangelopoulos, V. Giannakouris-Salalidis, L. Iliadis, C. Makris, Y. Plegas, A. Plerou, S. Sioutas, Evaluating information retrieval using document popularity: An implementation on MapReduce, Journal of Engineering Applications of Artificial Intelligence 51(2016), 16-23.

[17] V.K. Sharma, N. Mittal, Exploiting parallel sentences and cosine similarity for identifying target language translation, Journal of Procedia Computer Science 89(2016) 428-433.

[18] T. Xie, Q. Zheng, W. Zhang, A behavioral sequence analyzing framework for grouping students in an e-learning system, Journal of Knowledge-Based Systems 111(2016) 36-50.

[19] K. Seshadri, M. Shalinie S, Chidambaram Kollengode. Design and evaluation of a parallel algorithm for inferring topic hierarchies, Journal of Information Processing and Management 51(5)(2015) 662-676.

[20] M. Alewiwi, C. Orencik, E. Savaş, Efficient top-k similarity document search utilizing distributed file systems and cosine similarity, Cluster Computing 19(1)(2016) 109-126.

[21] N. Liu, P. Xiao, Y. Lu, Z.-J. Tang, H.-W. Wang, M.-X. Li, A topic approach to sentence ordering for multidocument summarization, in: Proc. the 14th IEEE International Symposium on Parallel and Distributed Processing with Applications, 2016.