Efficient Partial-Duplicate Detection Based on Sequence Matching

Qi Zhang, Yue Zhang, Haomin Yu, Xuanjing Huang
School of Computer Science, Fudan University
825 Zhangheng Road, Shanghai, P.R.China
{qi_zhang, 09210240052, 09210240086, xjhuang}@fudan.edu.cn

ABSTRACT

With the ever-increasing growth of the Internet, numerous copies of documents become serious problem for search engine, opinion mining and many other web applications. Since partial-duplicates only contain a small piece of text taken from other sources and most existing near-duplicate detection approaches focus on document level, partial duplicates can not be dealt with well. In this paper, we propose a novel algorithm to realize the partial-duplicate detection task. Besides the similarities between documents, our proposed algorithm can simultaneously locate the duplicated parts. The main idea is to divide the partial-duplicate detection task into two subtasks: sentence level near-duplicate detection and sequence matching. For evaluation, we compare the proposed method with other approaches on both English and Chinese web collections. Experimental results appear to support that our proposed method is effectively and efficiently to detect both partial-duplicates on large web collections.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval - Information Search and Retrieval; H.3.7 [**Digital Libraries**]: Collection, Systems Issues

General Terms

Algorithms, Experimentation.

Keywords

Partial-Duplicate Detection, Sequence Matching, MapReduce

1. INTRODUCTION

Because of the explosion of Internet and the fact that digital documents can be easily replicated, enormous duplicated web pages and mirrored documents cause serious problem for search engine, product review, and many other Web applications. Along with the increasing requirements, near-duplicate detection has received much attentions in recent years [24, 25, 11, 26, 20].

Existing studies on near-duplicate detection usually focus on the whole document level to figure out web pages that have the same content but only differ in the framing, navigation bar, advertisements, footer, and so on. Thus there are several factors that can not be well processed by existing methods.

Collection: Figure 1 shows a pair of Web pages[1] [2] which both of contain the article "Droid is No. 2 in Android traffic: Admob". Besides this article, the page in Figure 1.(a) contains another nine related ones. Thus, the similarity between the pages in Figure 1.(a) and (b) is low in the document level.

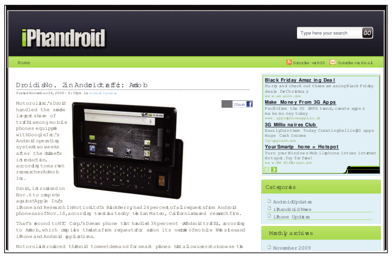
Multiple-page: In order to facilitate user's browsing, some articles are divided into multiple pages. Websites may use different strategies to split articles. Moveover, a number of websites may display the article in one page according to their own styles. It also leads to the similarities between the pages are low in document level.

Threads in Forum: Millions of people contribute more than 10 gigabytes content everyday through forums, blogs and other consumer-generated mediums [21]. However, user generated content often contains a couple of sentences/pragraphs copied from news sites or other users [14]. Since the duplications are usually only a small piece of text, they can not be effectively detected by existing methods.

Besides the factors listed above, there are a number of problems like, plagiarize sentences, non-cleaned web pages, sentences/paragraphs quotation, can also be generalized to *partial duplicate*. If a pair of documents are partial-duplicate with each other, it means they contain a number of sentences or paragraphs with similar content. With requirements of applications such as plagiarism detection, information flow tracking, opinion mining, and so on, partial-duplicate detection task is proposed and studied in this paper. Local text reuse detection [23] can be used to partially address this task. However, we argue that only similarities and category types do not provide sufficient information for all applica-

[1]http://iphandroid.com/
[2]http://www.chinapost.com.tw/business/company-focus/2009/11/25/234147/Droid-is.htm

tions and are not convenient enough for user to easily find the duplications in dozens of lines.



iPhandroid

Type here your search GO

Home | Subcribe via RSS | Subcribe via Email

Droid is No. 2 in Android traffic: Admob

Posted November 24, 2009 - 9:39pm in Android Updates

Share

Motorola Inc.'s Droid handled the second largest share of traffic among mobile phones equipped with Google Inc.'s Android operating system two weeks after the handset's introduction, according to market researcher Admob Inc.

Droid, introduced on Nov. 6 to compete against Apple Inc.'s iPhone and Research In Motion Ltd.'s BlackBerry had 24 percent of all requests from Android phones as of Nov. 18, according to estimates by the San Mateo, California-based research firm.

That's second to HTC Corp.'s Dream phone that handled 36 percent of Android traffic, according to Admob, which compiles the data from requests for ads on its network of mobile Web sites and iPhone and Android applications.

Motorola introduced the Droid to meet demand for smart phones that allow users to browse the

Black Friday Amazing Deal
Hurry and check out these amazing Black Friday deals for Christmas
www.amazon.com

Make Money From 3G Apps
Profit from the 3G APPS trend, create apps & make money today
www.applephoneapps.co.uk

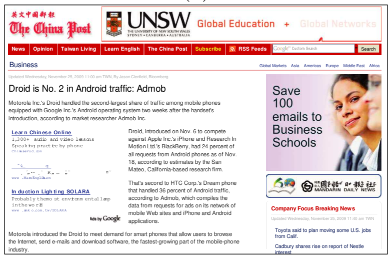
3G Millionaires Club
Easily Join them Today Creating/Selling 3G apps Huge Cash Income
3g-appcash.com

Your Smartphone = Hotspot
Turn your Windows Mobile phone into an internet Hotspot. Try for free!
www.WM WiFiRouter.com

Categories
Android Updates
iPhandroid News
iPhone Updates

Monthly archives
November 2009

(a)



英文中國郵報 The China Post

UNSW THE UNIVERSITY OF NEW SOUTH WALES SYDNEY • CANBERRA • AUSTRALIA Global Education + Global Networks

News | Opinion | Taiwan Living | Learn English | The China Post | Subscribe | RSS Feeds | Google Custom Search | Search

Business

Global Markets Asia Americas Europe Middle East Africa

Updated Wednesday, November 25, 2009 11:00 am TWN, By Jason Clenfield, Bloomberg

Droid is No. 2 in Android traffic: Admob

Motorola Inc.'s Droid handled the second-largest share of traffic among mobile phones equipped with Google Inc.'s Android operating system two weeks after the handset's introduction, according to market researcher Admob Inc.

Learn Chinese Online
1,300+ audio and video lessons Speaking practice by phone
ChinesePod.com

[illegible]
www.MarsEnglish.cn

Induction Lighting SOLARA
Probably the most environmental lamp in the world
www.anko.com.tw/SOLARA

Ads by Google

Droid, introduced on Nov. 6 to compete against Apple Inc.'s iPhone and Research In Motion Ltd.'s BlackBerry, had 24 percent of all requests from Android phones as of Nov. 18, according to estimates by the San Mateo, California-based research firm.

That's second to HTC Corp.'s Dream phone that handled 36 percent of Android traffic, according to Admob, which compiles the data from requests for ads on its network of mobile Web sites and iPhone and Android applications.

Motorola introduced the Droid to meet demand for smart phones that allow users to browse the Internet, send e-mails and download software, the fastest-growing part of the mobile-phone industry.

Save 100 emails to Business Schools

MANDARIN DAILY NEWS

Company Focus Breaking News
Updated Wednesday, November 25, 2009 11:40 am TWN
Toyota said to plan moving some U.S. jobs from Calif.
Cadbury shares rise on report of Nestle interest

(b)

Figure 1: Examples of partial-duplicate web pages

In this paper, we present an efficient algorithms for detecting partial-duplicates and locating their positions. Figure 2 shows an example on partial-duplicates. As shown in the graph, a sequence of sentences in Page A are similar with a number of sentences in Page B. Page A and C also contains duplicated text. From these pairs, we try to get the following results:

- Page A (Sen_i to Sen_j) $\backsim$ Page B (Sen_k to Sen_l).
- Page A (Sen_m to Sen_n) $\backsim$ Page C (Sen_p to Sen_q).

Since the proposed method can not only detect duplicates but also locate their positions, the near-duplicates of the whole document level can also be precisely detected. As the Web collections contain hundreds of millions pages, the algorithm is explored with MapReduce [8], which is a framework for large-scale distributed computing. We implement our method and compare it with the state-of-the-art approaches on four web collections and one manually constructed evaluation corpus. The experimental results show that it achieves good performance, both effectiveness and efficiency are significantly improved.

The contributions of this work are as follows: 1) We convert the partial-duplicate detection task into sentence level near-duplicate detection task and sequence matching task. 2) In order to handle hundreds of millions documents, the algorithm is designed and implemented under the MapReduce framework. 3) Shingles, I-Match, and Spotsigs are compared and evaluated in experiments, and experimental analyses of the signatures for sentences are provided. 4) Evaluations on manually labeled "Oracle Set" and four large web collections are used to measure the effectiveness and efficiency.

The remaining of the paper is organized as follows: In section 2, we review a number of related work and the state-of-the-art approaches in related areas. Section 3 provides an brief introduction of MapReduce. Section 4 presents the proposed method. Experimental results in test collections

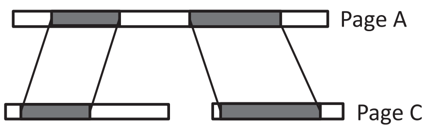


Figure 2: Partial-duplicate content

and analyses are shown in section 5. Section 6 concludes this paper.

2. RELATED WORK

Near-duplicate detection has received considerable attentions over the past several years. Previous studies on duplicate and near-duplicate detection can be roughly divided into two research directions: document representation and efficient detection. The first one focuses on representing documents with or without linguistic knowledge. Since collection contains hundreds of millions of documents, the second one, efficiency, has also received lots of attentions. This section introduces related approaches briefly.

Broder [3] defined the resemblance and containment between two documents. He used *shingles* to represent documents and Jaccard overlap to calculate the similarity between documents. In order to reduce the complexity of shingling, Broder [4] proposed to use meta-sketches for this task.

Indyk and Motwani[15] proposed the notion of locality-sensitive hashing and applied it to sublinear-time similarity searching. LSH maintains a number of hash tables, which each of is parameterized by the number of hashed dimensions. Points close to each other in some metric space have the same hash value with high probability. Gionis et al. [11] also used LSH for approximate similarity search.

I-Match [7] hinges on the premise that removal of very infrequent terms and very common terms results in good document representations for the near-duplicate detection task. They filter the input document based on collection statistics and compute a single hash value for the remainder text. The documents with same hash value are duplicates.

Schleimer et al. [22] proposed a local document fingerprinting algorithm, which is called winnowing. They described and analyzed the winnowing algorithm for selecting fingerprints from hashes of k-grams. They also presented the complexity of any local document fingerprinting algorithm and gave the non-trivial lower bound.

Henzinger [13] performed an evaluation of Border et al.'s [4] shingling and Charikar's [6] random projection near-duplicate algorithms on 1.6B web pages. The results showed that neither of the algorithms works well for detecting near-duplicate pairs on the same site, while both of them achieve high precision for near-duplicate pairs on different sites.

Manku et al. [19] proposed an approach for both online and batch types near-duplicate detection. They used Charikar's fingerprinting technique [6] and demonstrated it's effectiveness. They also presented an algorithmic technique for identifying existing f-bit fingerprints that differ from a given fingerprint in at most k bit-positions, for small k.

Theobald et al. [26] presented their work SpotSigs, which combine stopword antecedents with short chains of adjacent content terms. Through demonstrating the upper bounds of Jaccard similarity, they also proposed several pruning conditions, which could ignore all pairs of documents safely during the matching process when SpotSig vectors exceed a certain difference in length.

Besides the approaches focused on Web pages or documents, Muthmann et al. [20] proposed their work to identify threads with near-duplicate content and to group these threads in the search results. They incorporated text-based features, features based on extracted entities for products, and structure-based features to capture the near-duplicate threads.

Local text reuse detection proposed by Seo and Croft [23] is also related to our method. Different from duplicate detection, text reuse tries to capture the loose restatements of the information from the previous sources [2]. They defined six categories of text reuse and a general framework for text reuse detection. Several fingerprinting techniques for the framework were evaluated under the framework.

Lin [18] explored the problem of pairwise similarity on large document collections and introduced three MapReduce algorithms to solve this problem, which are based on brute force, large-scale ad hoc retrieval, and the Cartesian product of postings lists. Different with us, the granularity of this work is also document level.

Kolak and Schilit [16] described an approach to mine popularly quoted passages and add links among them on a digital library. They use shingle table method to find repeated sequences between different books. Since the storage complexity of shingle methods is huge and extracting shared shingles is timing consuming tasks, the method can not be directly used for partial-duplicate detection task.

In order to handle hundreds of millions web collections, we also use MapReduce framework in this work, which is introduced by Dean and Ghemawat [8]. It is used an associated implementation for processing and generating large data sets. The MapReduce programming model has been successfully used at Google for many different purposes.

3. MAPREDUCE

As number of data such as web pages, web request logs, and so on grows rapidly, applications have to be distributed across thousands of machines in order to finish in time. Bulk-synchronous parallel (BSP) model [27] and some higher-level abstractions(MPI [12]) have been supported programmers to write parallel programs. However, because of its higher-level abstractions, programmers usually spend too much time on details. MapReduce [8], which is difference from these systems, exploits a restricted programming model to parallelize the user program automatically. And the transparent fault-tolerance and load balancing are also provided, because of the restrictions.

The key concept behind MapReduce is inspired by the *map* and *reduce* primitives present in many functional languages. Dean and Ghemawat [8] presented the observation that most information processing computations share the same two-stage structure, which contains *map* and *reduce* operations. The *map* operation is applied to every logical "record" of input to compute a set of intermediate key/value pairs. Then the *reduce* operation is applied to all the values that shared the same key, in order to combine the derived data. Figure 3 shows the two-stage structure.

Under this framework, the computation takes a set of input key/value pairs, and produces a set of output key/value pairs. A programmer only needs to implement two operations: *map* and *reduce*. The intermediate key/value pairs will be grouped and sorted by the key automatically.

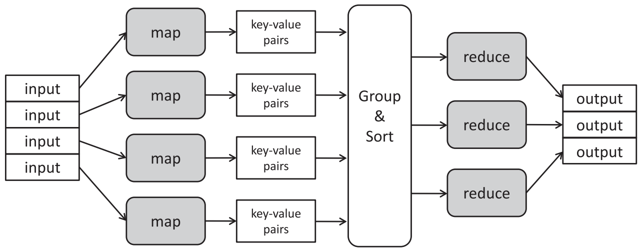


Figure 3: The basic structure of MapReduce

Many different implementations of MapReduce interface are available now. Google's MapReduce implementation is coupled with Google File System (GFS) [10], a kind of distributed file system. Apache's MapReduce implementation, Hadoop[3], which follows the same architecture, uses a distributed file system named Hadoop Distributed File System (HDFS) to store data and the intermediate results. Hadoop tries to schedule the MapReduce computation tasks to the node where the data locates in order to reduce the overall network I/O. Besides Hadoop, MapReduce has also been implemented by many corporations, such as Greenplum, GridGain, Cell Broadband Engine, and so on.

In this paper, we implement our algorithms under the open-source implementation Hadoop 0.20. HDFS is used to provide the distributed storage.

4. OUR APPROACH

A partial-duplicate is a pairwise relationship. Given a pair of documents, we need to identify and locate the duplicated parts between them. To make questions simple, we limit granularity to sentence level. Based on this assumption, we propose the algorithm ***PDC-MR***, which converts the partial-duplicate detection task into three MapReduce jobs (illustrated in Figure 4 and Figure 5).

1) Indexing: We use a MapReduce job to build a standard inverted index [9] for collections. Signatures used as terms in the inverted index are extracted from each sentences in map procedure. The map procedures emit the signature as the key, and a tuple consists of the document id and sentence id. After grouping and sorting, the reduce procedures take the tuples as input and write out the inverted index to the disk. Since signatures would highly impact the final result, a detail description of it will be given in the Section 4.1.

2) Sentence Duplication Detecting: Jaccard coefficient is used to measure the similarities between sentences. If the Jaccard similarity between a sentence pair is over a threshold, they are considered duplicates. Another MapReduce job is used to detect the sentence duplicates. The map procedures read the inverted index from disks and emit a pair of sentences which both contain a same signature as the key. After grouping and sorting, all signature ids belonging to the same sentence pair are brought together. The reduce procedures take them as inputs, and emit the sentence duplications. The procedure is shown in the right of Figure 4.

[3]http://hadoop.apache.org/

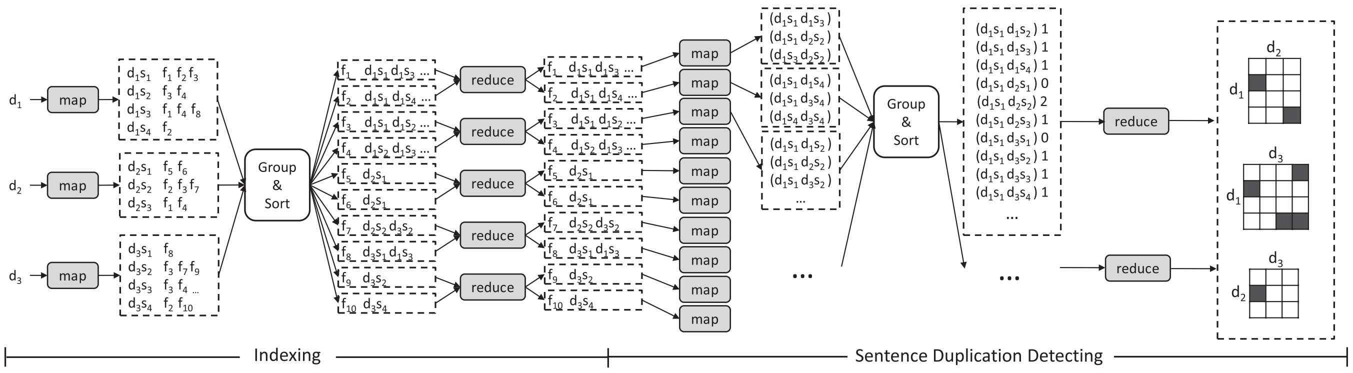


Figure 4: Detecting sentence duplication of a toy collection of 3 documents.

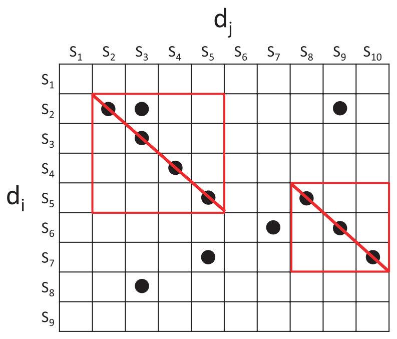


Figure 5: The sequence matching strategy

3) Sequence Matching: With the results of sentence duplicate detection, matrixes representing sentence duplicates for each pair of documents are generated. Figure 5 shows an example of the sentence duplicates between page d_i and page d_j. The dot plots in the figure represents duplicated sentence pairs. The sequences of duplicated sentences are partial duplications we try to extract and locate. Based on that, the problem can be straightly converted to the sequence matching task, which aims to find all diagonals in the matrix. We also use a MapReduce job to do that. The outputs of the job include partial duplicates among documents and their locations. Since numerous of document pairs are needed to be processed, Section 4.3 gives detail descriptions about the efficient sequence matching method.

4.1 Signatures

As described in the Section 2, a number of signature extraction methods have been proposed for document level near-duplicate detections. Since the average number of words per sentence is much fewer than document, we introduce several signature methods in this section.

4.1.1 *Shingles*

Shingles is the simplest method, which is proposed by Broder et al. [5]. It tokenizes documents into a list of words and extracts all word sequences of adjacent words to represent the document. "*n-shingles*" represents the number of n adjacent words in a shingle. As the shingles uses all chunks, it might not be suitable for large collections because of too many signatures.

4.1.2 *I-Match*

I-Match [7] uses SHA1 hash function over concatenation of terms filtered by stopwords and infrequent terms. It hinges on the assumption that removal of very infrequent terms and stop words results in good document representations for the near-duplicate detection task. Although the computationally of I-Match is attractive, it usually unstable even to small perturbations of content.

4.1.3 *SpotSigs*

SpotSigs [26] combines stopword antecedents with short chains of adjacent content terms. A spotsig s_i of a location in a document consists of a chain of words that follow an antecedent word a_i at a fixed spot distance d_i. Antecedent words are predefined and typically chosen to be stop words. Experimental results in [26] show that SpotSigs with five common terms as antecedent achieve better result than a full stopword list. However, we observe that signatures can not be extracted from more than 15.2% sentences in English collection with the five common terms. The experimental results about selecting the number of antecedents are shown in Section 5.3.

4.2 Sentence Duplication Detection

As shown in Figure 4, the sentence duplicate detection algorithm, which is implemented by a MapReduce job, extracts near duplicated sentence pairs whose Jaccard similarity are higher than a threshold. Sentences are represented by a group of signatures. The upper bounds for Jaccard similarity [26] is

$$J(A,B) = \frac{|A \bigcap B|}{|A \bigcup B|} \leq \frac{\min(|A|,|B|)}{\max(|A|,|B|)} \quad (1)$$

For $|A| \leq |B|$, we can get:

$$J(A,B) \leq \frac{|A|}{|B|} \quad (2)$$

With the upper bound and vector representation of documents, we observe that only similar length sentence pairs can be near duplicate. If we set the threshold to τ, sentence pairs where $\frac{|A|}{|B|} \leq \tau$ can be safely removed. Based on that, the pseudo-code of this method is show in Algorithm 1. The input of the procedure *map* is the signature id (sig_i) and associated postings list ($[d_1s_1, d_2s_2, ...]$, where d_is_j represents document id and sentence id). Inside each mapper, all candidate sentence pairs, which follow the upper bound of the Jaccard similarity, are emitted to the key-value pair $(\langle d_i s_j, d_k s_l \rangle, sig_i)$. After grouping and sorting, all signature ids belonging to the same sentence pair are brought together. With the list, Jaccard similarity can be easily calculated. The procedure *reduce* takes the sentence pair and corresponding list as input and emit the duplication judgments based on the Jaccard similarity and predefined threshold τ.

Algorithm 1 Pseudo-code of sentence duplication detection algorithm in MapReduce

```
MAP($sig_i, [d_1 s_1, d_2 s_2, ...]$)
1: for all $d_i s_j \in [d_1 s_1, d_2 s_2, ...]$ do
2:    for all $d_k s_l \in [d_1 s_1, d_2 s_2, ...]$ do
3:       if $d_i s_j \neq d_k s_l$ then
4:          if ( $|d_i s_j| \geq |d_k s_l|$ and $\frac{|d_k s_l|}{|d_i s_j|} \geq \tau$ )
            or ( $|d_i s_j| \leq |d_k s_l|$ and $\frac{|d_i s_j|}{|d_k s_l|} \geq \tau$ ) then
5:             EMIT($\langle d_i s_j, d_k s_l \rangle, sig_i$)
6:          end if
7:       end if
8:    end for
9: end for

REDUCE($\langle d_i s_j, d_k s_l \rangle, [sig_1, sig_2, ...]$)
1: if $\frac{|d_i s_j \bigcap d_k s_l|}{|d_i s_j \bigcup d_k s_l|} < \tau$ then
2:    EMIT($\langle d_i, d_k \rangle, \langle s_j, s_l \rangle$)
3: end if
```

4.3 Sequence Matching

As described in the previous sections, the sequence matching procedure aims to find all diagonals in the matrix. Algorithm 2 shows the pseudo-code of the MapReduce job. Inputs to the procedure *map* consists document pairs (keys, $\langle d_i, d_j \rangle$) and a corresponding list of duplicated sentence pairs between these documents (values, $[\langle s_k, s_l \rangle, \langle s_p, s_q \rangle, ...]$). For each duplicated sentence pair, the longest diagonal whose root is the pair is extracted and emitted. Extracted sentence pairs will be eliminated. γ is used as the threshold for the diagonal length. The final output, which contains document pair, respective start positions, and length, are generated in the procedure *reduce*. In practical, the reducer can also be merged into the mapper to trim the intermediate data.

Algorithm 2 Pseudo-code of sequence matching algorithm in MapReduce

```
MAP($\langle d_i, d_j \rangle, [\langle s_k, s_l \rangle, \langle s_p, s_q \rangle, ...]$)
1: $P \leftarrow [\langle s_k, s_l \rangle, \langle s_p, s_q \rangle, ...]$
2: for all $s_i s_j$ in $P$ do
3:    $D \leftarrow$ DiagonalExtract($s_i s_j$)
4:    if $|D| > \gamma$ then
5:       EMIT($\langle d_i, d_j \rangle, D$)
6:       $P \leftarrow P - D$
7:    end if
8: end for

DiagonalExtract($s_i s_j$)
1: while $s_i s_j$ in $P$ do
2:    $D \leftarrow D \bigcup s_i s_j$
3:    $s_i \leftarrow s_{i+1}$
4:    $s_j \leftarrow s_{j+1}$
5: end while

REDUCE($\langle d_i, d_j \rangle, [D_1, D_2, ...]$)
1: for all $D \in [D_1, D_2, ...]$ do
2:    EMIT($\langle d_i, d_j \rangle$, $Start.d_i, Start.d_j, D.Length$)
3: end for
```

5. EXPERIMENTS

5.1 Collections

We evaluate our methods with four corpora WT10g, TREC Blogs06[4], SogouT 2.0[5], and ClueWeb09-T09B[6]. Table 1 shows the statistics of the four collections. WT10g is used by TREC Web tracks, which contains more than 1.6 million documents collected from about 11,000 servers. Besides that, BLOGS06 corpus, which is used by TREC 2006 and TREC 2007 blog tracks, is also selected to evaluate systems. It is a big sample of the blogsphere, and contains more than 3.2 millions documents including spam as well as possibly non-blogs. SogouT 2.0 corpus is made up of 24.8M Chinese Web pages and crawled from all domains. TREC Category B dataset(ClueWeb09-T09B), which is a subset of the ClueWeb09, contains 50 million English pages and has been used in various TREC tracks.

Table 1: Statistics of the evaluation corpora

Corpus	Language	#Docs	Size
WT10g	English	1,692,096	11GB
Blogs06	English	3,215,171	88.8GB
SogouT 2.0	Chinese	24,833,521	372.5GB
ClueWeb09-T09B	English	50,220,423	490.4GB

5.2 Implementation and Setup

All the MapReduce jobs were implemented in Java for Hadoop framework. HDFS was used to provide the distributed storage. All experiments were evaluated on a 16 machines cluster. Each machine contains two Xeon quad core CPUs (2.0GHz), and 32GB RAM. Software stack of the experiments used Java 1.6 and Hadoop version 0.20. For web page cleaning, we just removed all HTML markup tags from the collections. Since the impact of sentence boundary detection's performance would not be heavy and a number of manually written rules can achieve good result [1] with little attractive computational consumption, we used around 50 rules to do that in our experiment.

5.3 Comparison of Signatures

In order to compare the performances of different signatures, we manually select 2000 documents, which contain 57135 sentences totaly, from ClueWeb09-T09B (*Oracle Eng* is used to represent the corpus in the following section for simple). For Chinese corpus SogouT, we also constructed a manually labeled corpus (*Oracle Chn*), which contains

[4]http://ir.dcs.gla.ac.uk/test_collections
[5]http://www.sogou.com/labs/dl/t.html
[6]http://boston.lti.cs.cmu.edu/Data/clueweb09/

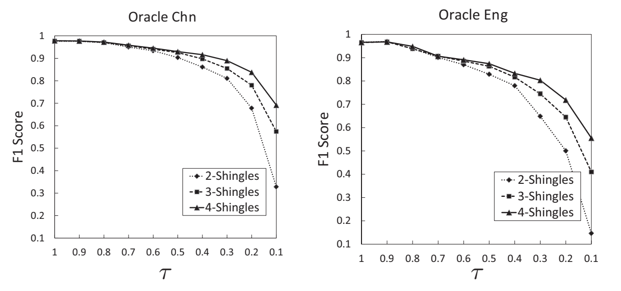


Figure 6: Shingles' performances of varying the threshold τ for corpora Oracle Eng and Oracle Chn

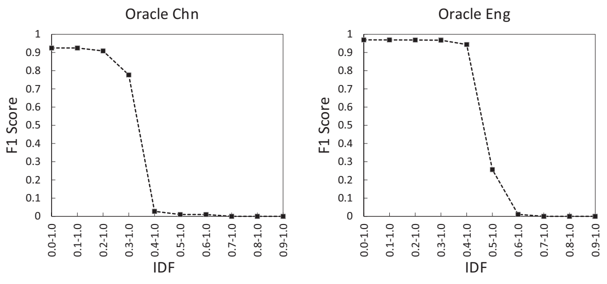


Figure 7: I-Match' performances of varying IDF for corpora Eng and Oracle Chn

80516 sentences extracted from 2000 documents. Six individuals were asked to label them. The average Kappa statistic among them is around 91.6%, which shows good agreement.

Figure 6 shows the performances comparison of 2-Shingles, 3-Shingles, and 4-Shingles. We observe that 4-Shingles consistently performs better than 2-Shingles and 3-Shingles in both English and Chinese collections. Different with results in document level [18, 26], threshold $\tau = 0.9$ achieves the best performance in both of the collections. The reason is that around 91% of duplicated sentences in Oracle Chn and 89% of them in Oracle Eng are exactly same with each other in our evaluation collections. However, this kind of factor is rare in the document level.

Figure 7 shows the performances of I-Match with different IDF ranges. Tokens exceeding IDF range were filtered. We use $idf_i = \frac{log(N/df_j)}{log(N)}$ to calculate the IDF value for token i, where N is the corpus size, df_j is the document frequency of the token. Since the similarities calculated by I-Match are either 0 or 1, the threshold τ does not need to adjusted. The best result is achieved by [0.1, 1.0] in both Oracle Chn and Oracle Eng. It means that most of the tokens should be kept and used to calculate the hash result. The main reason is that sentences usually contain a small number of tokens and most of the duplicated sentences are same with each other. When tokens whose IDF is lower than 0.4 are filtered, most of the sentences have less than 2 tokens left in Oracle Chn. Because of that, the recall for [0.4, 1.0] is almost perfect 100%, but the precision is only 1.4%.

The impacts of the number of antecedents for Spotsig are shown in Figure 8. The x-axis represents the number of antecedents and varies from 5 to 500 in Oracle Chn and 10 to 20K in Oracle Eng. The numbers below each point represent the average number of signatures per sentences with corresponding antecedents. It shows that the antecedents' number would highly impact the performance. We think that the main reason is that sentences cannot be well represented by a small number of signatures. By trading-off between efficiency and effectiveness, we choose # antecedent = 60 to achieve 95.2% F1 score in Chinese collection. For English one, we choose # antecedent = $10K$. We observe that the number of antecedents is much different between English and Chinese collections. However the best results are both achieved at the similar average number of signatures per sentence. It shows that a sentence can be well described by around 4 signatures. Figure 9 shows the performances with different thresholds. Comparing with shingles, spotsigs show the similar trends. We achieve the best result with $\tau = 0.9$ in Oracle Chn and Oracle Eng.

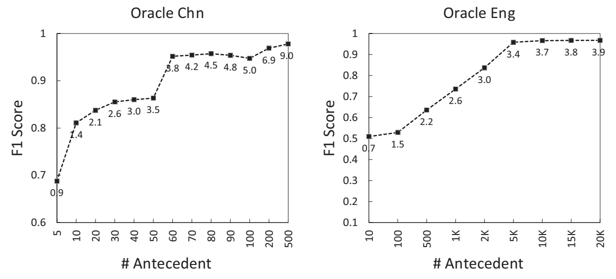


Figure 8: Spotsigs' performances of varying the number of antecedents for corpora Oracle Eng and Oracle Chn

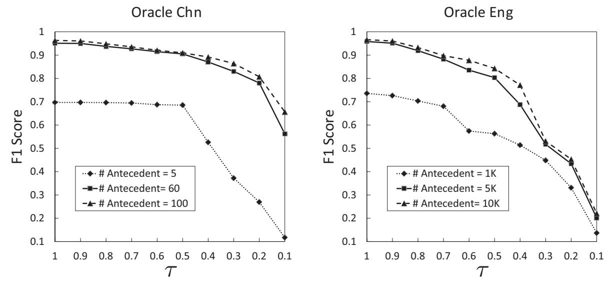


Figure 9: Spotsigs' performances of varying the threshold τ for corpora Oracle Eng and Oracle Chn

In summary, 4-Shingles achieve the best result in the sentence level duplicate detection. However, the performances of 2-Shingles, 3-Shingles, Spotsigs, and I-Match are comparable. The parameters used for sentence level are much different with document level ones. We think that it is caused by the characters of sentence collections, such as length, standard for labeling and so on. We also observe that although all three signature extraction methods are highly tunable, the results prove to be robust for a large variety of parameters.

5.4 Effectiveness Evaluation

After evaluating three different methods to extract duplicated sentences, we now consider the impact of sequence matching. Table 2 summaries the sequence matching results with different signatures. The configurable parameters IDF range, similarity threshold τ, and # antecedent are selected by the previous experiments and listed in the brackets. We use Precision, Recall, and F_1-Score as our choice of evaluation metric to measure how accurately the dupli-

Table 2: Summary of sequence matching results with Shingles, I-Match and Spotsigs for Oracle sets

Corpus	Signature	P	R	F1
Oracle Chn	2-Shingles($\tau = 0.9$)	0.936	0.937	0.936
	3-Shingles($\tau = 0.9$)	0.937	0.937	0.937
	4-Shingles($\tau = 0.9$)	0.942	0.942	0.942
	I-Match(IDF=[0.1,1.0])	0.935	0.938	0.937
	Spotsigs(#A=60, $\tau = 0.9$)	0.938	0.930	0.934
Oracle Eng	2-Shingles($\tau = 0.9$)	0.987	0.966	0.977
	3-Shingles($\tau = 0.9$)	0.987	0.966	0.977
	4-Shingles($\tau = 0.9$)	0.987	0.967	0.977
	I-Match(IDF=[0.1,1.0])	0.985	0.960	0.972
	Spotsigs(#A=10K, $\tau = 0.9$)	0.981	0.965	0.973

cation is located. From analyzing the Oracle collections, we observe that lengths of most duplications are bigger than three. Hence, γ, which is the threshold of diagonal length, is set to 3 in all the experiments. We observe that the final results are heavily related to the performances of sentence duplicate detection. Since the performances of 2-Shingles, 3-Shingles, 4-Shingles, I-Match and Spotsigs are similar, the final F_1-scores do not have significant difference. In order to evaluate the impact of γ, we also evaluate the performances at $\gamma = 1$. In Oracle Eng, the F_1-score of 2-Singles is only 0.952, which is significantly [7] different from the results shown in the Table 2. 3-Shingles, 4-Shingles, I-Match and Spotsigs have the similar results. By trading-off efficiency and effectiveness, we determine to use I-Match method to extract signatures in our method.

Figure 10 summarizes our results of PDC-MR versus the document level near-duplicate detection. For convenient comparison among copra, the top one million documents of each corpus are used in this experiment. For document level near-duplicate detection, the state-of-the-art method Spotsig is used, whose parameters are set up based on [26]. The y-axis represents the number of unique documents. The bottom parts of each bar represent results of Spotsig. The top parts represent the number of documents which can be detected by our PDC-MR method but can not be detected by document level Spotsig. In WT10g, Spotsig extracts around 31K documents which contain duplications in the same corpus. They compose more than 1.89 million duplication pairs. Besides those documents, through our method, another 94K documents which contain partial-duplicates are detected. In Blogs06, ClueWeb09-T09B, and Chinese corpus SogouT2.0, we get similar results. It shows that partial-duplications are common in web collections and our proposed method can effectively detect them.

In order to evaluate the validity of the extracted partial duplicates, we random select 200 documents from the detection results of each corpus and manually classify them into four types as listed in the Table 3. "News Collection" and "Multiple Page" are described in the Section 1. "Partial Quotation" represents all types of short piece of text quotation. Banner, copyright notice, navigation bar, and other non-content parts are classified into "Other". The results show that "Partial quotation" account for the majority of all instances. The average length of this kind of duplications

[7]The paired τ-test (ρ<0.05) is used to measure the significance.

Table 3: Partial duplicates in the web collections

Corpus	News Collection	Multiple Page	Partial Quotation	Other
WT10g	4%	8.5%	80%	7.5%
Blogs06	2.5%	5%	79%	13.5%
SogouT 2.0	10%	18%	60%	12%
ClueWeb09-T09B	3%	7%	58%	32%

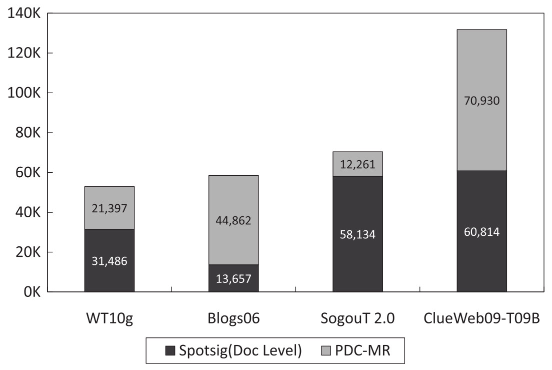


Figure 10: Summary of PDC-MR vs. document level Spotsig in four web collections

is around 6 sentences. While the average length of document is more than 23 sentences in SogouT 2.0 and 26 sentences in WT10g. Thus those partial duplications can not be easily detected by the existing document level detection methods. The results show that most of extracted partial duplications are useful and meaningful. Except ClueWeb09-T09B, the percentages of "Other" type in other collections are less than 15%. While, there are 32% instances belonging to this type in ClueWeb09-T09B. We think the main reason is that ClueWeb09-T09B is not well cleaned and contains lots of advertisements.

5.5 Efficiency Evaluation

Figure 11 plots the running times of spotsigs based near-duplicate detection and our proposed PDC-MR method for different corpus size. ClueWeb09-T09B is used in this experiment. All Hadoop jobs in the efficiency experiments were configured with 60 mappers and 60 reducers. The graph suggests that although the number of sentence is huger than the number of documents, our proposed method is more efficient than Spotsig. We think that it makes sense since I-match is efficient and its performance is also comparable in sentence level.

6. CONCLUSIONS

This paper presents our work on partial-duplicate detection task. A number of factors like news collection, multiple pages, threads in forums, plagiarize sentences, non-cleaned web pages, and sentences/paragraphs quotation belong to it. In order to address this problem, we propose a novel MapReduce algorithm, which converts the task into three MapReduce jobs. Except for the similarities between documents, the algorithm can simultaneously output the positions where the duplicated parts occur. The contributions of the work include both empirical analysis of signatures for

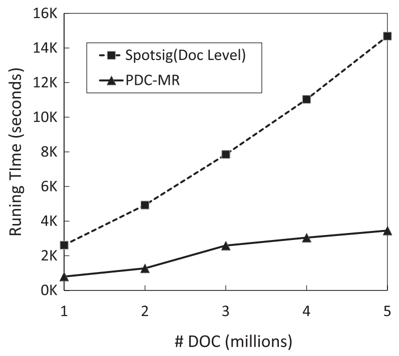


Figure 11: Running time of the PDC-MR and Spotsig with different corpus size

sentence and algorithm design. Experimental results in four real-world web collections show that the proposed method can be effectively and efficiently used to detect partial- and near-duplicate.

7. ACKNOWLEDGMENTS

The author wishes to thank the anonymous reviewers for their helpful comments. This work was partially funded by 973 Program (2010CB327906), Shanghai Leading Academic Discipline Project (B114), Doctoral Fund of Ministry of Education of China (200802460066), and Shanghai Science and Technology Development Funds (08511500302).

8. REFERENCES

[1] J. Aberdeen, J. Burger, D. Day, L. Hirschman, P. Robinson, and M. Vilain. Mitre: description of the alembic system used for muc-6. In *Proceedings of MUC6*, pages 141–155, Morristown, NJ, USA, 1995.

[2] M. Bendersky and W. B. Croft. Finding text reuse on the web. In *WSDM '09*, pages 262–271, New York, NY, USA, 2009. ACM.

[3] A. Z. Broder. On the resemblance and containment of documents. In *Proceedings of SEQUENCES 1997*, page 21, Washington, DC, USA, 1997. IEEE Computer Society.

[4] A. Z. Broder. Identifying and filtering near-duplicate documents. In *Proceedings of COM 2000*, pages 1–10, London, UK, 2000.

[5] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. *Comput. Netw. ISDN Syst.*, 29(8-13):1157–1166, 1997.

[6] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of STOC 2002*, pages 380–388, New York, NY, USA, 2002. ACM.

[7] A. Chowdhury, O. Frieder, D. Grossman, and M. C. McCabe. Collection statistics for fast duplicate document detection. *ACM Trans. Inf. Syst.*, 20(2):171–191, 2002.

[8] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of OSDI 2004*, San Francisco, CA, USA, 2004.

[9] W. B. Frakes and R. A. Baeza-Yates. *Information Retrieval: Data Structures & Algorithms.* Prentice-Hall, 1992.

[10] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. *SIGOPS Oper. Syst. Rev.*, 37(5):29–43, 2003.

[11] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB '99*, pages 518–529, San Francisco, CA, USA, 1999.

[12] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: portable parallel programming with the message-passing interface.* MIT Press, Cambridge, MA, USA, 1994.

[13] M. Henzinger. Finding near-duplicate web pages: a large-scale evaluation of algorithms. In *SIGIR '06*, pages 284–291, New York, NY, USA, 2006. ACM.

[14] S. C. Herring, L. A. Scheidt, I. Kouper, and E. Wright. A longitudinal content analysis of weblogs: 2003-2004. *Blogging, Citizenship and the Future of Media*, pages 3–20, 2006.

[15] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC '98*, pages 604–613, New York, NY, USA, 1998. ACM.

[16] O. Kolak and B. N. Schilit. Generating links by mining quotations. In *Proceedings of HT 2008*, pages 117–126, New York, NY, USA, 2008. ACM.

[17] A. Kołcz, A. Chowdhury, and J. Alspector. Improved robustness of signature-based near-replica detection via lexicon randomization. In *Proceedings of SIGKDD 2004*, pages 605–610, New York, NY, USA, 2004. ACM.

[18] J. Lin. Brute force and indexed approaches to pairwise document similarity comparisons with mapreduce. In *Proceedings of SIGIR '09*, pages 155–162, New York, NY, USA, 2009. ACM.

[19] G. S. Manku, A. Jain, and A. Das Sarma. Detecting near-duplicates for web crawling. In *WWW '07*, pages 141–150, New York, NY, USA, 2007. ACM.

[20] K. Muthmann, W. M. Barczyński, F. Brauer, and A. Löser. Near-duplicate detection for web-forums. In *IDEAS '09*, pages 142–151, New York, NY, USA, 2009. ACM.

[21] R. Ramakrishnan and A. Tomkins. Toward a peopleweb. *Computer*, 40(8):63–72, 2007.

[22] S. Schleimer, D. S. Wilkerson, and A. Aiken. Winnowing: local algorithms for document fingerprinting. In *SIGMOD '03*, pages 76–85, New York, NY, USA, 2003. ACM.

[23] J. Seo and W. B. Croft. Local text reuse detection. In *SIGIR '08*, pages 571–578, New York, NY, USA, 2008. ACM.

[24] N. Shivakumar and H. Garcia-Molina. Scam: A copy detection mechanism for digital documents. In *Digitial Library*, 1995.

[25] N. Shivakumar and H. Garcia-Molina. Finding near-replicas of documents and servers on the web. In *Proceedings of WebDB 1998*, pages 204–212, London, UK, 1999. Springer-Verlag.

[26] M. Theobald, J. Siddharth, and A. Paepcke. Spotsigs: robust and efficient near duplicate detection in large web collections. In *SIGIR '08*, pages 563–570, New York, NY, USA, 2008. ACM.

[27] L. G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, 1990.