Efficient Near-Duplicate Detection for Q&A Forum

Yan Wu, Qi Zhang, Xuanjing Huang
Fudan University
School of Computer Science
{10210240075,qz,xjhuang}@fudan.edu.cn

Abstract

This paper addresses the issue of redundant data in large-scale collections of Q&A forums. We propose and evaluate a novel algorithm for automatically detecting the near-duplicate Q&A threads. The main idea is to use the distributed index and Map-Reduce framework to calculate pairwise similarity and identify redundant data fast and scalably. The proposed method was evaluated on a real-world data collection crawled from a popular Q&A forum. Experimental results show that our proposed method can effectively and efficiently detect near-duplicate content in large web collections.

1 INTRODUCTION

There is a rise in popularity of Question and Answering forums in recent years. The forums allow users to post, browse, search and answer questions. Q&A forum acts not only as a medium for knowledge sharing , but also as a place in which one can seek advice, and satisfy others' curiosity about a countless number of things (Adamic et al., 2008). However, because of the ever-increasing growth of it, and the fact that users are not always experts in the areas they post threads on, duplicate content becomes a serious issue. And, most of the current Q&A forums haven't had a efficient mechanism to identify threads with near-duplicate content. As a consequence, users have to go through different versions of duplicate or near-duplicate content, and are often frustrated by it. Baidu Zhidao[1] is one of the largest Q&A forums in China. It contains more than 100 million question and answer pairs. Because of the increasing popularity and the number of users, it also encounters the long-standing problem – content duplication. For example, there are more than four hundred question-answer pairs which contain the same content "When is the birthday of Jay?" Other Q&A forums are facing the similar problem.

Along with the increasing requirements and the limitations of manual methods, there have been growing research activities in duplicate detection, during the past few years. Common automatical methods to detect duplicates are copy detection or near-duplicate detection (Gionis et al., 1999; Muthmann et al., 2009; Shivakumar and Garcia-Molina, 1995; Shivakumar and Garcia-Molina, 1999; Theobald et al., 2008; Zhang et al., 2010). Most of the current near-duplication detection approaches usually focus on the document level to figure out the web pages with different framing, navigation bar, and advertisements, but duplicate content. However, unlike duplicated web-pages, forum threads have the following differences:

1. Forum threads contain additional structured meta information, e.g. title, tags, external/internal links, etc. So, the method used to detect near-duplicate of the forum thread is to the web-page.

2. The average length of threads is usually less than the news articles. Adamic et al. reported that most of the thread lengths are less than 400 words in Yahoo! Answers(Adamic et al., 2008).

3. The number of threads grows in a significant pace compare to other media. Thus a more efficient method need to be used to handle millions of content.

[1] http://zhidao.baidu.com

Proceedings of the 5th International Joint Conference on Natural Language Processing, pages 1001–1009,
Chiang Mai, Thailand, November 8 – 13, 2011.

In this paper we propose a novel algorithm for detecting near-duplicate threads in the Q&A forums. Threads with similar content can be identified. The proposed algorithm completes the pairwise similarity comparisons in two steps: inverted index building and then similarity computations with it. Thread content and other meta information can be represented by signature/feature sets. As the Web collections contain hundreds of millions pages, this algorithm is done through MapReduce (Dean and Ghemawat, 2004), which is a framework for large-scale distributed computing. We implement our method and compare it with the state-of-the-art approaches on a real-world data crawled from a Q&A web forum and one manually labeled evaluation corpus. From the experimental results, we can observe that both effectiveness and efficiency are significantly improved. The major contributions of this work are as follows:

- We analyze the common structure of threads in Q&A forums, and give a definition of near-duplicate thread.
- We propose efficient solutions, which use distributed inverted index and are implemented under Map-Reduce framework, for calculating pairwise similarity and identifying redundant data fast and scalably.
- We describe a number of signatures for unstructured content and other meta information, and experimentally evaluate them.
- A tight upper bound of Jaccard coefficient is given and used in the to speed up the similarity calculation.
- We evaluate our method on a real-world corpus crawled from Baidu Zhidao. Experimental results showed that our algorithm can achieve better result than the state-of-the-art algorithms for detecting near-duplicate web pages on forum content.

The rest of the paper is structured as follows: In Section 2, a number of related work and the state-of-the-art approaches in related research areas are briefly described. Section 3 defines the problem we try to deal with and gives the introduction of MapReduce framework. In Section 4, we present the proposed methods. Experimental results in testing collections and performance evaluation are shown in Section 5. Finally, Section 6 concludes this paper.

2 RELATED WORK

Near-duplicate detection has been widely studied over the past several years. Previous works on duplicate and near-duplicate detection can be roughly divided into two research ares: document representation and efficiency. The first one focuses on how to represent a document with or without linguistic knowledge. The second area, which focuses on how to handle hundreds of millions of documents, has also received lots of attentions. The technique of estimating similarity among pairs of documents was presented by Broder et al. (Broder, 1997). They used *shingles*, which does not rely on any linguistic knowledge, to represent documents and Jaccard overlap to calculate the similarity. I-Match (Chowdhury et al., 2002) divided the duplicate detection into two tasks: 1) filtering the input document based on collection statistics; 2) calculating a single hash value for the remainder text. The documents with same hash value are considered as duplicates. SpotSigs was proposed in 2008 by Theobald et al. (Theobald et al., 2008), which combines stopword antecedents with short chains of adjacent content terms. Hajishirzi et al. (Hajishirzi et al., 2010) presented an adaptive near-duplicate detection method, which can achieve high accuracy across different target domains. Besides the approaches focused on Web pages or documents, Muthmann et al. (Muthmann et al., 2009) proposed their work to identify threads with near-duplicate content and to group these threads in the search results.

3 PRELIMINARIES

Although there are several different websites which provide the Q&A service, their question-answer thread structure are very similar. Usually, each thread includes a question (Title), none or several sentences used to describe the question (*Description*), a best an-

swer, and a number of other answers. Based on the common structure of Q&A forum threads, we will use the following definition to capture near-duplicate threads:

Definition 1 *Near-Duplicate Thread – Two threads are near-duplicate with each other in Q&A forum: (1) if both of their question and answer parts are the same with each other, or (2) if their question parts are same and one of the answers contains additional information compare to another answer.*

Consider the following examples:

Example 1
Question: *When is the birthday of Jay?*
Description: The birthday of Jay.
Best Answer: January 18th, 1979

Example 2
Question: *Who can tell me what's the Jay's birthday?*
Description: Jay "male, top R&B singer", I want to know when he was born.
Best Answer: 18/1/1979

Since the question part of example 1 and example 2 have the same meaning and answer parts are also the same, they are near-duplicate threads according to our definition, although the two threads use different words and expressions in the question parts and different date formats in the answer parts.

4 OUR APPROACH

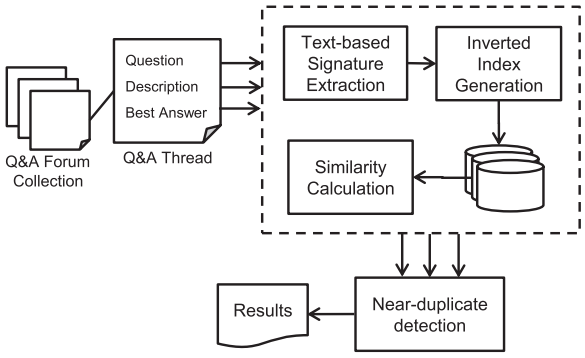

Figure 1: Process for detecting near-duplicate threads

Figure 1 shows the process for identifying near-duplicate threads in Q&A forums. The process consists of four stages:

Stage 1. Text-based Signature extraction produces signatures for each thread. Signatures for different parts are separately extracted. We only consider three parts including "Question", "Description", and "Best Answer" in this paper.

Stage 2. Inverted index generation treats signatures as terms and builds distributed inverted indexes for collections.

Stage 3. Similarity Calculation solves the pairwise similarity comparison problem with the generated distributed inverted indexes.

Stage 4. Near-duplicate detection identifies near-duplicate threads based on the calculated similarities among Q&A threads in different parts.

In Stage 1, we try to use several types of signatures extracted from different meta-information to partially overcome the problem of word-overlap limitation. The efficient pairwise similarity comparison problem is captured in Stage 2 and 3. This section describes two algorithms for solving the similarity calculation problem with two kinds of distributed inverted index: *Term-based Index* and *Doc-based Index*. These two algorithms are described in turn. Both term-based and doc-based algorithms follow the unified framework. Based on the similarities among different parts, the near-duplicate detection is done in Stage 4.

4.1 Upper Bound of Jaccard Similarity

Jaccard coefficient is widely used to measure the similarities among sets. In this work, we use it to measure the similarities among forum threads, which are represented by a group of signatures. $J(A,B) = \frac{|A \bigcap B|}{|A \bigcup B|}$ is the default Jaccard similarity defined for two sets. Theobald et al. described the bounds of it (Theobald et al., 2008), which is

$$\begin{aligned} J(A,B) &= \frac{|A \bigcap B|}{|A \bigcup B|} = \frac{|A \bigcap B|}{|A| + |B| - |A \bigcap B|} \\ &\leq \frac{\min(|A|,|B|)}{\max(|A|,|B|)} \end{aligned} \quad (1)$$

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

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

With the upper bound and vector representation of threads, we observe that near-duplicated threads have the similar length. If

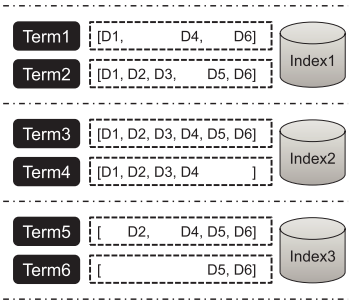

Figure 2: Example of term-based distributed index

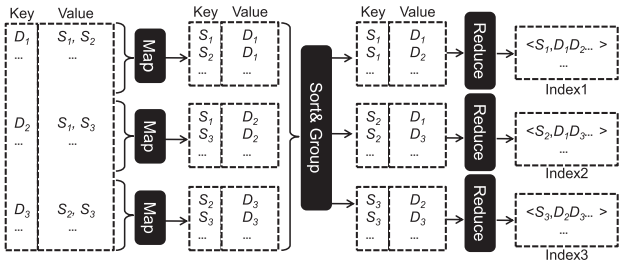

Figure 3: Data flow of term-based distributed index generation

we set the threshold to τ, thread pairs where $\frac{|A|}{|B|} \leq \tau$ can be safely removed.

Inspired by Eq.2, we further propose a more tightly upper bound of Jaccard similarity as follows:

$$J(A,B) = \frac{|A \bigcap B|}{|A \bigcup B|} = \frac{|A| - |A-B|}{|B| + |A-B|} \leq \frac{|A| - |C|}{|B| + |C|}, C \subseteq A - B \quad (3)$$

It can be easily proofed that this bound 3 is tighter than the upper bound of Eq. 2. If the set C can be detected, more calculation can be reduced with it. In this work, both of the upper bounds are used to reduce the size of intermediate data. In algorithm 1, the Eq.2 is used. While the more tightly bound Eq.3 is used in the algorithm 2.

4.2 Term-based approach

Inverted index is an index data structure storing a mapping from terms (signatures in this work) to its documents. In order to distributedly calculate the pairwise similarity, the inverted index should be split into multiple parts, which can be parallelly processed in the further steps. The term-based distributed index splits the inverted index according to the rows. Each partition of the distributed index only contains a number of terms and their corresponding posting list. Figure 2 shows an example of it. In the figure, "Index 1" contains posting lists of "Term 1" and "Term 2". Other terms are separately stored in "Index 2" and "Index 3".

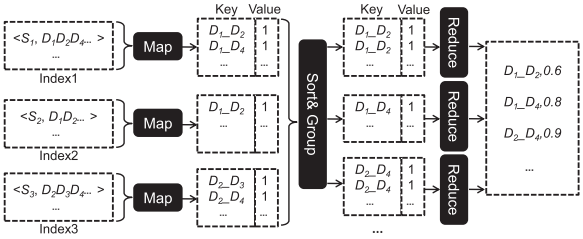

Figure 4: Data flow of similarity calculation based on term-based distributed index

Algorithm 1 Pseudo-code of term-based algorithm

MAP$(S_i, [D_1, D_2, ...])$

1: **for all** $D_i \in [D_1, D_2, ...]$ **do**
2: **for all** $D_j \in [D_1, D_2, ...]$ **do**
3: **if** ($|D_i| \geq |D_j|$ **and** $\frac{|D_j|}{|D_i|} \geq \tau$) **or** ($|D_i| \leq |D_j|$ **and** $\frac{|D_i|}{|D_j|} \geq \tau$) **then**
4: EMIT$(\langle D_i, D_j \rangle, S_i)$
5: **end if**
6: **end for**
7: **end for**

REDUCE$(\langle D_i, D_j \rangle, [S_1, S_2, ...])$

1: **if** $\frac{|D_i \bigcap D_j|}{|D_i \bigcup D_j|} \geq \tau$ **then**
2: EMIT$(\langle D_i, D_j \rangle)$
3: **end if**

The data flow of term-based distributed index generation using MapReduce is shown in Figure 3. Input to the distributed indexer consists of thread ids as key and extracted signatures as terms. In each mapper operation, all signatures are iteratedly processed. For each signature, a pair consisting of the signature id as key and the thread id as value is created. The mapper emits those key-value pairs as intermediate data. After grouping and sorting, thread ids which contain the same signature are grouped together. The reducer gets a part of key-value pairs (term and associate posting

list) as input and emits them as a partition of the distributed index.

Based on the term-based distributed index, the data flow of near-duplicate detection algorithm is shown in Figure 4. The input of the procedure *map* is the signature id (S_i) and associated postings list ($[D_1, D_2, ...]$, where D_i represents thread id). Inside each mapper, all candidate thread pairs which fit the the Eq. 2 , the upper bound of the Jaccard similarity, are emitted to the key-value pair ($\langle D_i, D_j \rangle, S_i$). After grouping and sorting, all signature ids belonging to the same thread pair are brought together. With the list, Jaccard similarity can be easily calculated. The procedure *reduce* takes the thread pair and corresponding list as input and emits the Jaccard similarity (the predefined threshold τ is used to reduce the size of output). The pseudo-code of this algorithm is shown in Algorithm 1. The line 4 in the algorithm is based on the upper bound of Jaccard similarity and used to reduce the size of intermediate data.

However, in practical terms, the term-based method can not be directly used to process the collection which contains more than one million threads. Too many intermediate outputs will cause problem. In this work, we limit the maximum number of intermediate output as the approximation method, which is proposed by Lin (Lin, 2009). This approximation can improve the efficiency of the term-base algorithm, based on which the term-based method can process large collections. However the calculated similarities may not reflect the real similarities with this approximation method.

4.3 Doc-based approach

Different to the term-based distributed index, doc-based distributed index splits the inverted index according to the columns. Each partition of doc-based distributed index only contains a number of threads and their corresponding terms. Figure 6 shows an example of it. In the figure, "Index 1" contains terms which are contained in the "D_1" and "D_2"and their corresponding posting lists.

Figure 5 shows two kinds of doc-based distributed index generation data flows using MapReduce. Input to both of the distributed indexers is the same as the term-based one, which consists of thread ids as key and extracted signatures as terms. Figure 5(a) shows a standard inverted index generation procedure using MapReduce. Different with the standard one, only a partition of forum threads collection are input to the MapReduce job at each time. And the procedure is iterated over the entire collection. An alternative method is shown in Figure 5(b). The entire collection is processed with a MapReduce job. In each mapper operation, a partition of doc-based distributed index is generated and is written to the disk. No reducers are required in the job. Although the second approach is simple and no iteration is required, the size of the input collection is much smaller compare to the first one, because of the memory limitation.

Algorithm 2 Pseudo-code of doc-based algorithm

MAP$(S_i, [D_1, D_2, ...])$

1: **for all** $D_i \in InitializedIndex$ **do**
2: **for all** $D'_i \in InputIndex$ **do**
3: **if** ($|D_i| \geq |D'_i|$ **and** $\frac{|D'_i|}{|D_i|} \leq \tau$) **or** ($|D_i| \leq |D'_i|$ **and** $\frac{|D_i|}{|D'_i|} \leq \tau$) **then**
4: **continue**
5: **end if**
6: $C \leftarrow \emptyset$
7: $\delta \leftarrow 0$
8: **for all** $S_{ij} \in D_i$ **do**
9: **if** $S_{ij} \in D'_i$ **then**
10: $\delta \leftarrow \delta + 1$
11: **else**
12: $C \leftarrow C \bigcup S_{ij}$
13: **if** $\frac{|D_i|-|C|}{|D'_i|+|C|} \leq \tau$ **then**
14: **continue**
15: **end if**
16: **end if**
17: **end for**
18: $sim(D_i, D'_i) \leftarrow \frac{\delta}{|D_i|+|D'_i|-\delta}$
19: EMIT$(\langle D_i, D'_i \rangle, sim(D_i, D'_i))$
20: **end for**
21: **end for**

REDUCE$(\langle D_i, D'_i \rangle, sim(D_i, D'_i))$

1: EMIT$(\langle D_i, D'_i \rangle)$

The data flow of similarity calculation algorithm based on the doc-based distributed

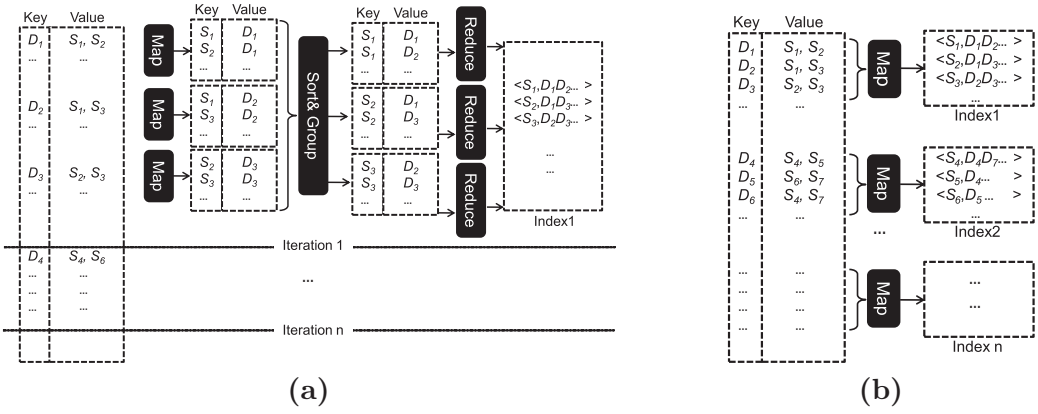

Figure 5: Data flow of doc-based distributed index generation

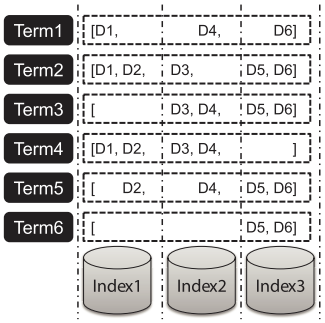

Figure 6: Example of doc-based distributed index

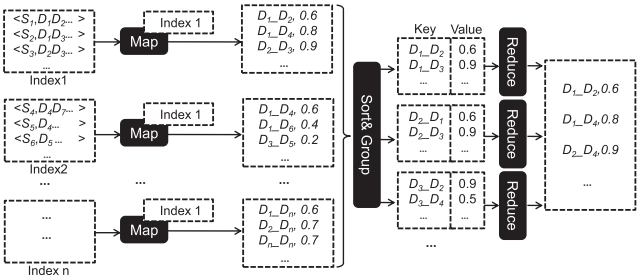

Figure 7: Data flow of one iteration of the similarity calculation based on doc-based distributed index

index is shown in Figure 7. The whole procedure should be iterated until all partitions of the distributed index have been processed. At the each iteration, all mappers are initialized with a same partition of distributed index (E.g. "Index 1" in the figure). The input of the procedure *map* is the signature id (S_i) and associated postings list ($[D_1, D_2, ...]$, where D_i represents thread id). Different mappers will get different partitions of the index as input. Inside each mapper, a inverted-index based algorithm is used to calculate the similarities between the initialized index and the input index. The pseudo-code of this algorithm is shown in Algorithm 2.

4.4 Near-Duplicate Detection

Based on the definition of near-duplicate thread, the final stage, near-duplicate detection, tries to combine the similarities calculated through the previous steps and other information extracted from threads to determine whether threads are near-duplicate or not. Obviously, most of the popular machine learning methods (e.g. Support Vector Machines, Maximum Entropy, AdaBoost, and so on) can be used to solve the problem with the calculated similarities and extracted information as feature sets. However, for simplification, in this paper we use linear combination of different parts' similarities and a predefined threshold τ to do that. If the similarities $S(T_i, T_j)$ calculated through the Eq. 4 are bigger than τ, those Q&A pairs are emitted as near-duplicates.

$$S(T_i, T_j) = \theta_t S_T(T_i, T_j) + \theta_d S_D(T_i, T_j) + \theta_a S_A(T_i, T_j), \quad \text{where } \theta_t + \theta_d + \theta_a = 1, and\ \theta_t, \theta_d, \theta_a \geq 0 \tag{4}$$

S_T,S_D, and S_A respectively represent similarities calculated through different parts. θ_t, θ_d, and θ_a represent the weights of similarities between different parts, and are roughly tuned based on a small number of manually labeled corpus.

Table 1: Summarization of F_1 scores of different signatures in different parts

Signature	Question	Description	Answer
2-Shingles	59.33%	**99.54%**	90.74%
3-Shingles	**60.29%**	**99.54%**	**95.05%**
4-Shingles	58.33%	**99.54%**	91.58%
Winnowing($k = 3, w = 3$)	52.06%	97.24%	85.30%
Winnowing($k = 5, w = 3$)	52.62%	97.16%	87.69%
I-Match([0.10, 0.90])	53.59%	95.31%	44.16%
I-Match([0.20, 0.80])	51.82%	93.95%	44.95%
SpotSigs(# Antecedent=50)	20.77%	13.70%	74.56%
SpotSigs(# Antecedent=100)	34.06%	18.75%	75.54%

5 EXPERIMENTS

In this section, we detail experimental evaluations of the proposed method on both effectiveness and efficiency. We crawled a portion of the Baidu Zhidao, resulting in a local archive of about 28.6 million questions, all of which have user-labeled "best answers". The corpus covers more than 100 categories. From this set we manually selected 2000 question-answer threads as "Gold-Standard". Four individuals were asked to label the duplications among question-answer threads. Meanwhile, the duplications between threads in different parts have also been manually judged. The average kappa statistic among them is around 73.2%, which shows good agreement.

We ran experiments on a 15-node cluster. Each node contains two Intel Xeon processor E5430 2.66GHz with four cores, 32GB RAM, and 128GB hard disk. We used an extra node for running the master daemons to manage the Hadoop job and distributed file system. Software stack of the experiments used Java 1.6 and Hadoop version 0.20.2. All the MapReduce jobs were implemented in Java for Hadoop framework. HDFS was used to provide the distributed storage.

5.1 Comparison of Signatures

In order to compare the performance of different signatures in the Q&A domain, the "Gold-Standard" serves in the following experiments, since these near-duplicates have been manually judged by humans. Performance comparison of different signatures with various parameters in question, description, and answer parts is shown in this section.

Table 1 summarizes the best result of different signatures in question, description and answer parts (the main parameters are shown in the bracket). We observe that 3-Shingles achieve the best result in all three parts. However, the performances of 2-Shingles, 4-Shingles are comparable. I-Match and SpotSigs achieve worse result than other methods. The possible reasons are given in the previous of the section. We also observe that although all signature extraction methods are highly tunable, the results are stable with a large range of parameters' values. With all different signatures, the performance of question part is not very satisfactory. After carefully examining the results, we observe that questions with same meaning often differ significantly in syntax and language. In (Jeon et al., 2005; Muthmann et al., 2009), it is also mentioned that two threads that have the same meaning may use different wording. Most of the signatures used for near-duplicate detection can not process this kind of issue very well. We think that it is also the main reason of the low performance of I-Match. The best F_1 score of question part is only 60.29%, however, the precision can achieve 96.46%. It indicates that although not all the duplications can be detected, most of the duplications we extracted are right. For answer part, since the average length is more than 200 characters, the performance of it is satisfactory.

5.2 Performance of Near-Duplicate Detection Stage

To investigate the parameters used in the Eq.3 and the threshold τ, hill climbing algorithm is used for tuning the four parameters. We randomly selected 100 initial seeds. Table 2 shows a number of F_1 score with different parameters. The best F_1 score of the near-duplicate thread detection is 66.22% (P=94.98%, R=50.83%) in this domain with parameters $\theta_t = 0.4$, $\theta_t = 0.2$, $\theta_t = 0.4$, and $\tau = 0.5$. Based on the definition of near-duplicate threads, question parts should have the same intuition. Because of this, the final detection performance is highly impacted by the performance of question part. More natural language processing techniques would improve the detection recall. While it may also consume much more computational time.

5.3 Term-based V.S. Doc-based

To judge and compare the efficiency and scalability of doc-based and term-based distributed

Table 2: Performance of near-duplicate detection stage with different parameters

θ_t	θ_d	θ_a	τ	P	R	F_1-Score
0.2	0	0.8	0.1	10.73%	**96.90%**	19.32%
0.3	0.6	0.1	0.1	17.30%	69.01%	27.66%
0.5	0.3	0.2	0.2	24.82%	62.40%	35.51%
0.4	0.5	0.1	0.2	35.88%	54.34%	43.22%
0.7	0.1	0.2	0.3	46.49%	57.44%	51.39%
0.6	0.2	0.2	0.3	65.01%	56.82%	60.64%
0.5	0.3	0.2	0.3	77.26%	54.75%	64.09%
0.4	0.2	0.4	0.5	**94.98%**	50.82%	**66.22%**
0.4	0.3	0.3	0.4	89.93%	51.65%	65.62%
0.4	0.4	0.2	0.3	40.50%	53.31%	46.03%
0.6	0	0.4	0.5	80.24%	54.55%	64.94%

index, it is best to evaluate them in a real-world data set. Figure 8 summaries the efficiency comparison between the two distributed indexing methods. 3-Shingles method is used to extract signatures from all three parts of the input corpus. In near-duplicate detection stage, parameters is set as follows: $\theta_t = 0.4$, $\theta_t = 0.2$, $\theta_t = 0.4$, and $\tau = 0.5$. The x-axis represents the size of the corpus, the total number of processing time is represented in y-axis. The doc-based method is much efficient than the term-based method when the size of the corpus is smaller than 2 million. Because of the approximation used in the term-based method, the slope of the term-based method's curve is lower than doc-based method ones when the size of the corpus is bigger than 2 million. However, the approximation also made the impact on the number of near-duplicates extracted by term-based method to be much lower than the doc-based method.

Figure 9 shows the number of detected near-duplicate threads through term-based method versus doc-based method. The total number of threads which are near-duplicate with one or more threads is shown in the figure. We observe that there are about 0.473 million (15.78%) threads which can be found one or more near-duplications in the corpus. We observe that although the corpus we used in this experiment only contain 3 million threads, 4.25 million of near-duplicate threads pairs are

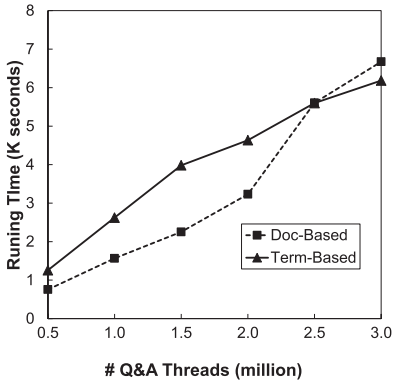

Figure 8: Term-based V.S. Doc-based in Efficiency

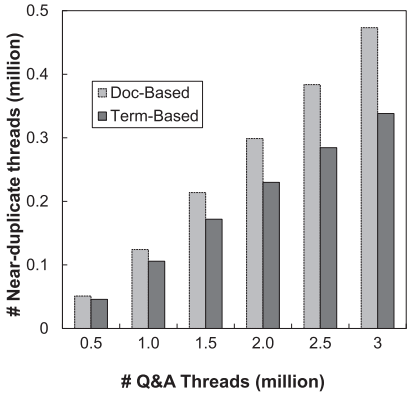

Figure 9: Number of detected Near-duplicated threads

extracted from it. It means that some popular questions have a huge number of near-duplicated ones.

6 CONCLUSION AND FUTURE WORK

In this paper we studied the problem of near-duplicate detection for Q&A forums. We proposed two distributed inverted index methods to calculate similarities in parallel using MapReduce framework. We defined the near-duplicate Q&A thread and used the evaluated signatures, parallel similarity calculating and a liner combination method to extract near-duplications. Experimental results in the real-world collection show that the proposed method can be effectively and efficiently used to detect near-duplicates. About 15.78% of Q&A threads contain more than one near-duplicates in the collection.

Acknowledgments

The author wishes to thank the anonymous reviewers for their helpful comments. This work was partially funded by 973 Program (2010CB327906), National Natural Science Foundation of China (61003092, 61073069), Shanghai Science and Technology Development Funds(10dz1500104), Doctoral Fund of Ministry of Education of China (200802460066), Shanghai Leading Academic Discipline Project (B114), and Key Projects in the National Science & Technology Pillar Program(2009BAH40B04).

References

Lada A. Adamic, Jun Zhang, Eytan Bakshy, and Mark S. Ackerman. 2008. Knowledge sharing and yahoo answers: everyone knows something. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 665–674, New York, NY, USA. ACM.

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

Abdur Chowdhury, Ophir Frieder, David Grossman, and Mary Catherine McCabe. 2002. Collection statistics for fast duplicate document detection. *ACM Trans. Inf. Syst.*, 20(2):171–191.

Jeffrey Dean and Sanjay Ghemawat. 2004. Mapreduce: Simplified data processing on large clusters. In *Proceedings of OSDI 2004*, San Francisco, CA, USA.

Aristides Gionis, Piotr Indyk, and Rajeev Motwani. 1999. Similarity search in high dimensions via hashing. In *VLDB '99*, pages 518–529, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Hannaneh Hajishirzi, Wen-tau Yih, and Aleksander Kolcz. 2010. Adaptive near-duplicate detection via similarity learning. In *SIGIR '10: Proceeding of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 419–426, New York, NY, USA. ACM.

Jiwoon Jeon, W. Bruce Croft, and Joon Ho Lee. 2005. Finding similar questions in large question and answer archives. In *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 84–90, New York, NY, USA. ACM.

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

Klemens Muthmann, Wojciech M. Barczyński, Falk Brauer, and Alexander Löser. 2009. Near-duplicate detection for web-forums. In *IDEAS '09: Proceedings of the 2009 International Database Engineering & Applications Symposium*, pages 142–151, New York, NY, USA. ACM.

Narayanan Shivakumar and Hector Garcia-Molina. 1995. Scam: A copy detection mechanism for digital documents. In *Digitial Library*.

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

Martin Theobald, Jonathan Siddharth, and Andreas Paepcke. 2008. Spotsigs: robust and efficient near duplicate detection in large web collections. In *SIGIR '08*, pages 563–570, New York, NY, USA. ACM.

Qi Zhang, Yue Zhang, Haomin Yu, and Xuanjing Huang. 2010. Efficient partial-duplicate detection based on sequence matching. In *SIGIR '10: Proceeding of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 675–682, New York, NY, USA. ACM.