Efficient Duplicate Detection on Cloud Using a New Signature Scheme

Chuitian Rong[1,2], Wei Lu[1,2], Xiaoyong Du[1,2], and Xiao Zhang[1,2,3]

[1] Key Labs of Data Engineering and Knowledge Engineering, MOE, China
[2] School of Information, Renmin University of China
[3] Shanghai Key Laboratory of Intelligent Information Processing
{rct682,uqwlu,duyong,xiaozhang}@ruc.edu.cn

Abstract. Duplicate detection has been well recognized as a crucial task to improve the quality of data. Related work on this problem mainly aims to propose efficient approaches over a single machine. However, with increasing volume of the data, the performance to identify duplicates is still far from satisfactory. Hence, we try to handle the problem of duplicate detection over MapReduce, a share-nothing paradigm. We argue the performance of utilizing MapReduce to detect duplicates mainly depends on the number of candidate record pairs. In this paper, we proposed a new signature scheme with new pruning strategy over MapReduce to minimize the number of candidate record pairs. Our experimental results over both real and synthetic datasets demonstrate that our proposed signature based method is efficient and scalable.

Keywords: duplicate detection, MapReduce, Cloud.

1 Introduction

Duplicate detection is widely used in a variety of applications, including deduplication[1], data cleaning and record linkage [2][3]. Given a set of records[1], the objective of duplication detection is to identify records that refer to the same real-world entity. Due to a variety of quality problems, such as typos, abbreviations, update anomalies and combination thereof, two records that do not exactly match with each other may refer to the same entity. Two records whose similarity is not less than a pre-defined threshold are considered as duplicates. The straightforward approach to identify duplicates over a dataset requires the comparison on each pair of strings. Obviously, the computational cost will be extremely high when the dataset is large scale. As such, instead of offering each pair of strings as a candidate record pair, previous work mainly focuses on proposing efficient approaches, including indexing techniques such as inverted index[4], Trie Tree [5] and B^+-Tree[6] to reduce the number of candidates pairs. However, with increasing volume of data, the performance of these approaches is still far from satisfactory especially when the dataset cannot be properly reside in the main memory.

[1] In this paper, for each record, we concatenate its attribute values into a single string.

H. Wang et al. (Eds.): WAIM 2011, LNCS 6897, pp. 251–263, 2011.

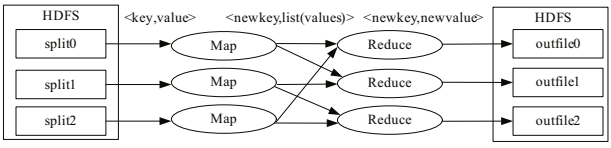


Fig. 1. Overview of the MapReduce Execution

In order to alleviate the problem that incurred by the increasing scale of datasets, we try to apply the MapReduce paradigm to detect the duplicates. MapReduce is a programming paradigm for processing large-scale datasets over a share-nothing parallel cluster. Fig.1 shows its execution overview. It consists of two procedures: Map and Reduce. In this framework, key/value pairs are sorted and shuffled based on their keys; the data items with the same key will be grouped together. In case of duplicate detection by using MapReduce, the key work is to assign the duplicate candidates the same key, here we say signature. Then, the duplicate candidates can be shuffled into one group and sent to one Reduce node for last verification. The signature can be defined as the common characteristics of duplicates candidates. It must have discriminative power, that is to say duplicate candidates must have same signature and records with different signatures can not be duplicates. We argue that the performance of detecting duplicates in MapReduce framework mainly depends on the number of candidate set size in each group. To achieve better performance, new proper signature scheme and pruning strategy are proposed to decrease the pseudo duplicates in each candidate set.

As the MapReduce program paradigm is a share-nothing framework. There are two challenges need to be solved. One, we must apply proper signature scheme for duplicate candidates, which has direct effect on duplicate candidate size and the efficiency of verification. As the data is split and the task on different nodes is done independently, there is no more global information can be used. The other challenge is that we must devise new prune method to decrease the pseudo duplicates as many as possible in new environment. Hence, in this paper, we make the following contributions:

- We proposed solution for duplicate detection on cloud by using new signature scheme. The signature scheme can produce exact signatures for each candidate record pairs.
- We proposed a new pruning strategy to minimize the pseudo candidates number in each candidate set.
- Redundant comparisons for the same candidate record pairs are eliminated in our solution.

The rest of the paper is organized as follows: Section 2 presents the problem definitions and preliminaries. Section 3 describes our proposed signature scheme. Next, duplicate detection using this signature scheme is illustrated in Section 4.

Experiments study is given in Section 5.Related work is covered in Section 6 and Section 7 concludes this paper.

2 Problem Definition and Preliminaries

2.1 Problem Definition

Definition 1. ***n-gram token set*** *Let r be a string. A set of n-gram tokens of r denoted as $G_r = \{g_1^r, g_2^r, g_3^r, ...\}$,where g_i^r is defined as follows. $g_i^r = r[i-n+1, i-n+2, ..., i]$, where $r[s, ..., t]$ means the substring of r from s^{th} to t^{th}. If s is less than 0 or t is greater than the length of r, then the blank is replaced by special character. So, $|G_r| = |r| + n - 1$.*

Example 1. For example, record "Jim Gray" can be tokenized as a set of 3-grams: { ##j, #Ji, Jim, im , m G, Gr, Gra, ray, ra$, a$$ }.

As there does not exist a similarity function can always perform the best than other similarity functions in all application scenarios. In this paper, we explored a widely used similarity function Jaccard. Unless otherwise specified, $sim(r_i, r_j)$ refers to $sim_{Jaccard}(r_i, r_j)$.

Definition 2. ***Similarity*** *Given a record pair (r_1, r_2), let $G_{r_1} = \{g_1^{r_1}, g_2^{r_1}, ...\}$, $G_{r_2} = \{g_1^{r_2}, g_2^{r_2}, ...\}$ be the n-gram token set of r_1 and r_2, repectively. The similarity of record pair (r_1, r_2) is defined as $sim_{Jaccard}(r_1, r_2)$.*

$$sim_{Jaccard}(r_i, r_j) = \frac{|G_{r_i} \cap G_{r_j}|}{|G_{r_i} \cup G_{r_j}|} \tag{1}$$

Using the defined similarity function to evaluate the similarity between any two records,we can formally define the result of duplicate detection: $\{\langle r_i, r_j\rangle | \forall r_i, r_j \in R, sim(r_i, r_j) \geq \theta\}$, where θ is pre-assigned by users.

For the sake of brevity, we give some necessary symbols and their definitions in Table 1 that will be used throughout this paper.

2.2 Properties of Jaccard Similarity Function

Definition 3. ***Global Ordering*** *U is the universe of n-gram in data set R.*

$$\forall g_i, g_j \in U, \text{ if } tf_{g_i} \leq tf_{g_j} \text{ then } g_i \preceq g_j.$$

Definition 4. ***Prefix Tokens*** *For record r, $G_r = \{g_1^r, g_2^r, ..., g_{|G_r|}^r\}$. The tokens g_i^r in G_r are sorted by global ordering. Its prefix token is denoted as G_r^p.*

$$G_r^p = \{g_i | g_i \in G_r, 1 \leq i \leq |G_r| - \lceil |G_r| * \theta \rceil + 1\}.$$

Definition 5. *θ* ***Split Point*** *The θ split point is the token that ranked at $\lceil |r_i| * \theta \rceil - 1$ to the rear of record's sorted token list. The tokens that ranked before this token are the candidates for signatures.*

Table 1. Symbols and their definitions

Symbol	Definition
R	collection of records
r	record in R
U	finite universe of n-grams
G_r	sequence of n-grams for record r
$\|G_r\|$	the number of n-grams in G_r
G_r^p	the first p n-grams for record r
$\|G_r^p\|$	the number of n-grams in G_r^p
g	an n-gram of U
tf_g	term frequency of gram g
θ	pre-assigned threshold
$sim(r_i, r_j)$	the similarity between record r_i, r_j
$\preceq$	partial order

tokens	oij	094	loi	94#	iji	idy	jic	dyn	434	tme	c_3	stm	343	.....
term frequency	2	3	4	5	7	8	9	13	13	16	20	20	21	.....

Fig. 2. Prefix Tokens

Example 2. Take the record r="Advance duplicate address detect" for example and set $\theta = 0.8$. So,$|G_r|$=34 and $|G_r^p|$=$34 - \lceil 34 * 0.8 \rceil + 1 = 7$. This is shown in Fig.2.

When gram tokens in each record are sorted by global ordering, the Jaccard similarity function has the property[7].

Property 1. $\forall r_i, r_j \in R$, let $G_{r_i}^p$ and $G_{r_j}^p$ be their prefix token set, respectively. If $sim(r_i, r_j) \geq \theta$, then $|G_{r_i}^p \cap G_{r_j}^p| \geq 1$.

As the above property of Jaccard similarity, if two records are duplicates they must have one common prefix token at least. If each of prefix tokens of the record is used as its signature, the duplicate records can have the chance to be shuffled into one group in MapReduce framework. Then, we can verify them.

3 Signature Scheme

The signature scheme can be considered as blocking or grouping process, in which the duplicate candidates will be assigned the same signature. In MapReduce, the signature is used as the key for duplicate candidate records,then they can be shuffled into one group and sent to one Reduce node for the last verification. As MapReduce is a share-nothing programming paradigm, there is no more global information about records can be used as in traditional solution. The property of Jaccard similaity function can be used to generate signature. The direct approach is to use each prefix token as its signature in[8] and take each word as a token.

Algorithm 1. *SignatureGeneration*

```
1  Map(key,value):
2  newvalue ← key;
3  foreach item_i ∈ value do
4      select item_j from value using LengthPruning;
5      newkey ← (item_i, item_j)(i > j);
6      write(newkey,newvalue);
7  Reduce(key,values):
8  newvalue ← values;
9  TokenNumber ← the token number in newvalue;
10 result ← ReducePrune(key,TokenNumber);
11 if result==true then
12     write(key,newvalue);
```

When using the n-gram based method for duplicates detection, it is not effective. There are three causes:(1)One single prefix token may be also the prefix token of many other records;(2)Many record pairs have more than one common prefix token;(3)The number of gram tokens for the same record will be many times of its word number. All these can be seen from the Example3.

Example 3. Take these two records r_1 and r_2 below for example.

- r_1: Advanced duplicate address detect
- r_2: Advance duplicate address detection

Both r_1 and r_2 contain 4 words, but will produce 35 and 37 3-gram tokens respectively. Given threshold 0.8, if taking each word as a token they will produce only one prefix token, $\{Advanced\}$ and $\{Advance\}$ respectively[8]. If they are used as signature, these two records can't be shuffled into one group as candidates. So, taking each word as a token is sensitive to many typo errors. When split the record into 3-gram, their prefix tokens number are all 8. If each of gram in prefix tokens is used as the record's signature, there will be 8 signatures for each record. The common prefix tokens of them are $\{dup, add, ddr,\ du, dre, upl, dva, Adv\}$. If using the signature as in [8], these two records will be verified eight times. If just use single prefix token as signature, many redundancies and redundant verifications will be imported. As each prefix token of one record will be prefix token of many other records. Usually, many records will have more than one common prefix tokens.

By thorough analysis we proposed a new signature scheme using the records common prefix tokens. It is an exact signature scheme. During the signature generation we applied two prune strategies using record's properties.

3.1 Signature Generation

In order to produce the new signature we have to complete the works:(1)get a global ordering of all tokens in the data set;(2)build the inverted list just for

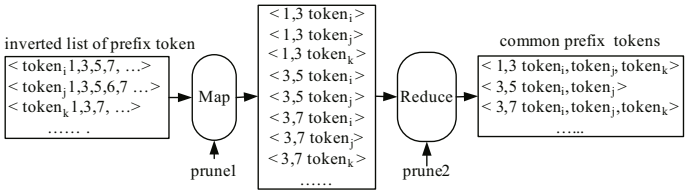


Fig. 3. Data Flow for Signature Generation

each distinct prefix token using MapReduce. Each item in inverted list contains the information about the record, such as record id, record length and its θ split value(term frequency). The item in inverted list is sorted by the value of record length in it; (3)get the maximum common tokens for each duplicate pair by analyzing the inverted list using pruning strategies. As build the inverted index is common place,we just give the details of signature generation.

This process is only based on the information of records, not the original record. The details are given in Algorithm1 and the data flow in this MapReduce procedure is shown in Fig.3, record length and θ split value information is omitted for illustrating clearly. In this procedure, the inverted index files are also split and processed by different Map. Each input key/value pair in the Map is an sorted inverted list for one prefix token(line 1). To get the longest common prefix tokens of each record pair, we set the input *key* to the *newvalue*(line2). Scan the inverted list from start using the length pruning strategy(line 3-6) and write out the two items that satisfy the pruning condition as *newkey*. In Reduce, the prefix tokens with the same key are common prefix tokens of two records(line 8). Then it apply the second pruning method(line 10). If it satisfies our requirement the tokens in *newvalue* is their signature.

3.2 Pruning Method

In order to improve the efficiency of signature generation and decrease the number of pseudo duplicates, we proposed two pruning methods used in Map and Reduce respectively. The first pruning method is used for each input line in Map. That is an length filtering method as shown in Algorithm 1(line 8). The second pruning method is used in Reduce, as shown in Algorithm 1(line 19),*ReducePrune*. As the length filtering is widely used,we don't illustrate here[9]. Below, we give our pruning strategy used in Reduce. It is just based on record length,common prefix token number and θ split values. As all tokens in the data set are sorted by global ordering, each token can be seen as a dot in the number axis. We use number axis and set properties to prove its correctness. Assume their common token number is K, spt_i and spt_j are their θ split values. In Fig.4, we can see there are two situations for these two records, $spt_i <= spt_j$ and $spt_i > spt_j$.

Lemma 1. *For $spt_i <= spt_j$,the prune condition is:*

$$\frac{K+\lceil |G_{r_i}|*\theta \rceil -1}{|G_{r_i}|+|G_{r_j}|-(K+\lceil |G_{r_i}|*\theta \rceil -1)} \geq \theta$$

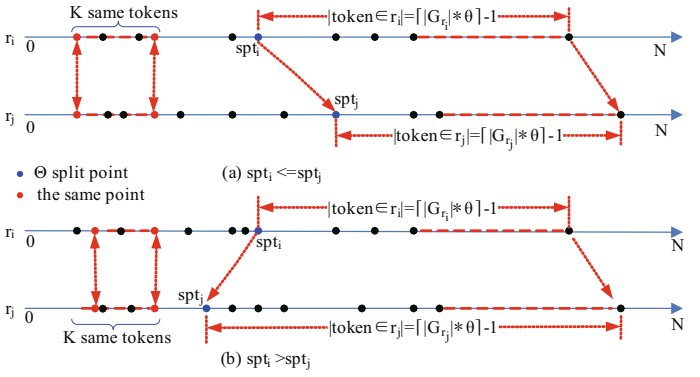


Fig. 4. Pruning Condition

Proof. The common tokens of these two records are come from two parts in the number axis, one part is before their split point spt_i and spt_j , the other part is after these two split points as shown in Fig.4.

The signature is their common tokens that come from the first part, the number is K. The maximum number of common tokens that come from the other part is determined by the length of two records and their θ split value. As $spt_i \leq spt_j$, the maximum number of common tokens from the last part is $\lceil |G_{r_i}| * \theta \rceil - 1$. So,the maximum common tokens for these two records is $K + \lceil |G_{r_i}| * \theta \rceil - 1$. If they are duplicates, the similarity of them can not less than the given θ. So the pruning condition is: $\frac{K+\lceil |G_{r_i}| * \theta \rceil - 1}{|G_{r_i}|+|G_{r_j}|-(K+\lceil |G_{r_i}| * \theta \rceil - 1)} \geq \theta$

Lemma 2. *For $spt_i > spt_j$,the prune condition is:*

$$\frac{K+\lceil G_{r_j} * \theta \rceil - 1}{|G_{r_i}|+|G_{r_j}|-(K+\lceil G_{r_j} * \theta \rceil - 1)} \geq \theta$$

As the proof is the same to the last, we don't give here for length limit of paper.

4 Duplicate Detection Processing

After the signature generation we have got the exact signatures for each duplicate candidate pair. Using the exact signature, the duplicate detection can be completed efficiently. The whole process is given in Algorithm 2. Before doing duplicate detection we distributed the signature index file as distributed cache file to each computing node[10]. In setup procedure, each computing node will use this file to build signature structures in its local main memory. In order to save space, each node just read in the related block of cache file about the processed data split. In the Map, for each input record its signatures can be get from signature structures that built in setup(line 3-6). The records are assigned their

Algorithm 2. *DuplicateDetection*

```
1  Setup:
2  read cache file and build signature information;
3  Map(key,value):
4  Signatures ← GetSignatures(value);
5  foreach sig ∈ signatures do
6      write(sig,value);
7  Reduce(key,values):
8  foreach val_i, val_j ∈ values do
9      sim ← sim_jaccard(val_i, val_j);
10     if sim ≥ θ then
11         newvalue ← (val_i, val_j);
12         write(key,newvalue);
```

signatures, then will be shuffled by their signatures and send to Reduce node for last verification. In Reduce, each pair of records with the same signature will be verified using similarity function $sim_{jaccard}(r_i, r_j)$(line 9-11).

Intuitively,when the token number in signature is more the records with this signature is less. Using this signature scheme can remove redundant verifications and can decrease candidate set size. We can prove the candidate set size produced by using common prefix token is far less than using single prefix token. Lets the candidates number of using single prefix token and using common prefix token is N_1,N_2 respectively. We can prove $N_2 \ll N_1$.

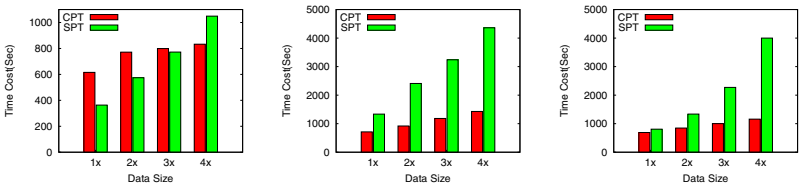
Proof. For direct approach, $\forall r_i \in R$, its prefix tokens is $G^p_{r_i}$. As, $\forall g^{r_i} \in G^p_{r_i}$ will be considered as signature and assigned to the record. So, the total intermediate records is $N_1 = \sum_{i=1}^{n} |G^p_{r_i}|$. As some prefix tokens just occur in one record, so $N_1 > 2\sum_{i=1}^{n}\sum_{j>i}^{n}\left|G^p_{r_i} \bigcap G^p_{r_j}\right|$.

For our proposed new signature scheme, $\forall r_i, r_j \in R$, it takes $G^p_{r_i} \bigcap G^p_{r_j}$ as their signature. If $G^p_{r_i} \bigcap G^p_{r_j} ! = \Phi(i \neq j)$, the result will be assigned to the record pair as signature, otherwise no signature is generated. The intermediate records number is $N_2 = 2\left|\left\{G^p_{r_i} \bigcap G^p_{r_j} \left| G^p_{r_i} \bigcap G^p_{r_j} ! = \Phi(i \neq j)\right.\right\}\right|$. As many records in the large scale data set have more than one common prefix tokens, $N_2 \ll N_1$.

5 Experimental Evaluation

5.1 Experiment Setup

We study the performance of duplicate detection using Hadoop platform over nine computer nodes, among which one is set to the master and the others are set to the slaves. Each node is equipped with 2 cores, 2G main memory, and the operating system is Ubuntu. The maximum tasks of Map and Reduce are set



(a) author (b) title (c) synthetic

Fig. 5. Comparison

to 40, the other configurations are their default values. For this evaluation, we utilize three datasets (including two real datasets and one synthetic dataset):

- Titles: It consists of 1,772,301 titles that are extracted from DBLP[2] and CiteSeer[3] with total size of 98M;
- Authors: It consists of 2,065,229 author names that are extracted from DBLP and CiteSeer with total size of 43M;
- Synthetic: It consists of 1,349,955 strings that are produced by Febrl[4] with total size of 91M.

We use the following algorithms in the experiments.

- **SPT** serves as the baseline algorithm [8], where for each record r, every gram in the prefix of G_r is taken as a signature. Hence, for every two records that share multiple grams in their prefixes, multiple record pairs will be produced as the candidates;
- **CPT** is our proposed algorithm, where for every two records r_i, r_j, the longest common grams of G_{r_i} and G_{r_j} are offered as the signature. Hence, for every two records that share multiple grams in their prefixes, only one record pair is produced.

Unless otherwise specified, the threshold that we use to detect duplicates in this paper is set to 0.9.

5.2 Signature Scheme Evaluation

In this section, we study the function of both CPT and SPT over the above three datasets. We scale each dataset by copying every record for multiple times that varies from 1× to 4×.

Fig.5 plots the cost of detecting duplicates using CPT and SPT that apply different signature schemes. From Fig.5(b)(c), as expected, CPT outperforms SPT. To better understand this point, we plot the distribution of longest common

[2] http://www.informatik.uni-trier.de/∼ley/db
[3] http://citeseerx.ist.psu.edu
[4] http://sourceforge.net/projects/febrl

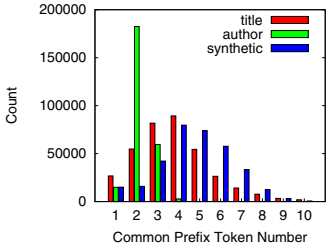


Fig. 6. Common Prefix Token Distribution

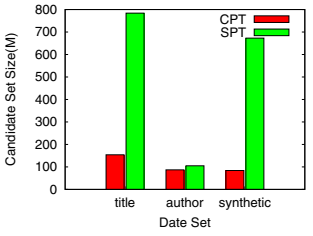
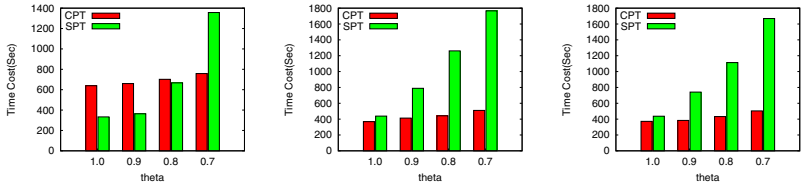


Fig. 7. Candidate Set Size



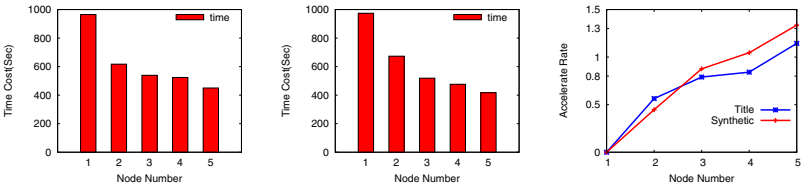
(a) author (b) title (c) synthetic

Fig. 8. Effect of threshold

prefix tokens of each candidate record pairs in Fig.6. As we can see that, over each dataset, there exist a large number of record pairs that share more than one common prefix tokens, redundant record pairs will be produced by applying SPT method. Hence, these redundant record pairs will be shuffled and verified meaninglessly. We show the size of candidate set for both SPT and CPT in Fig.7. Statistic information that is shown in Fig.6 and Fig.7 is extracted from the 1× dataset. Interestingly, SPT performs better than CPT over the author dataset when the size is small. As the record length in author data is short, the common prefix number for record pairs are almost between 2 to 3. So,when the data set size is small the cost for SPT method is smaller than CPT. When scaling the size of each dataset, the cost of both CPT and SPT increases. However, there is an obvious trend that SPT increases significantly while CPT increases smoothly.

5.3 Effect of Threshold

In this section, the effect of threshold is evaluated on 1× datasets. When the threshold decreases, prefix token number of each record and candidate pairs increase.So, the time cost increases. Fig.8(b) and (c) show that when the threshold decreases, the time cost of CPT increased smoothly, but the cost for SPT increased significantly. As threshold decreases, more candidates pairs are produced using SPT than using CPT. The special situation occurs in Fig.8(a) can be explained as in the last experiment.



(a) Title (b) Synthetic Data (c) Accelerate Rate

Fig. 9. Effect of Node Number

5.4 Effect of Node Number

The function of computing node number in the cluster is studied in this section. As the slave nodes in our cluster are heterogeneous, we chose five slaves that have same architecture to testify the function of node number. The 1× title and synthetic datasets are used. We can see from Fig.9, the all time cost is decreased when the nodes number is increased. In Fig.9(c), we give an figure about accelerate rate. It is to demonstrate the relative improvement of performance when using more than one computing nodes contrast to using one node. We can see as the node number increased the value is increased, but it is not linearly increased as the data transfer cost is also increased. Overall, when the slave number is increased the process performance is improved.

6 Related Work

The general duplicate detection problem has been also known as merge-purge[11], record linkage[2][3], object matching, reference reconciliation[12],deduplication and approximate string join, etc. All these works are the core of well-known data cleaning primitives. In order to improve performance, existing approaches usually have two phases: candidate generation and verification[13][14]. In different work the candidate generation phase can be seen as signature assignment process, its goal is to put the candidates into one group by using different strategies, including BKV[3], SNN[15] and Hashing based strategy[16][17]. When used in MapReduce to process large scale data, these methods are not effective as used in one machine. The Hashing based signature scheme is used in[18][19]. In [20], the prefix based signature scheme is proposed and used by[8] for similarity join in MapReduce. It can be seen as an exact kind of signature method. It used the token in one record with the smallest frequency as the signature, but this method will cause many redundant candidates and verifications, especially when the record is long. In [8], it takes words in record as tokens and uses single prefix tokens as signature. It is sensitive to many typo errors. Further, when tokenized the record into n-gram tokens set, many redundant candidate pairs will be produced. To address these problems, we proposed a new signature scheme using common prefix tokens of each candidate pair and a new pruning strategy to minimize the pseudo duplicates in MapReduce framework.

7 Conclusions and Future Work

In this paper, the MapReduce framework is used for duplicate detection. New signature scheme and pruning strategies are proposed. Experiments demonstrate that they are effective to decrease the candidate set size and performance improvement. In the future, we will do further work on other signature schemes and pruning strategies using different strategies.

Acknowledgements. This work is partly supported by National 863 High Technology Program (No. 2009AA01Z149), the Important National Science & Technology Specific Projects of China ("HGJ" Projects, No.2010ZX01042-002-002-03), the Fundamental Research Funds for the Central Universities (the Research Funds of Renmin University of China, No.10XNI018), Shanghai Key Lab. of Intelligent Information Processing, China(No. IIPL-09-018).

References

1. Arasu, A., Ré, C., Suciu, D.: Large-scale deduplication with constraints using dedupalog. In: ICDE, pp. 952–963 (2009)
2. Fellegi, I., Sunter, A.: A theory for record linkage. Journal of the American Statistical Association 64(328), 1183–1210 (1969)
3. Gu, L., Baxter, R., Vickers, D., Rainsford, C.: Record linkage: Current practice and future directions. CSIRO Mathematical and Information Sciences Technical Report 3, 83 (2003)
4. Xiao, C., Wang, W., Lin, X., Shang, H.: Top-k set similarity joins. In: ICDE, pp. 916–927 (2009)
5. Wang, J., Feng, J., Li, G.: Trie-join: Efficient trie-based string similarity joins with edit-distance constraints. PVLDB 3(1-2), 1219–1230 (2010)
6. Zhang, Z., Hadjieleftheriou, M., Ooi, B., Srivastava, D.: Bed-tree: an all-purpose index structure for string similarity search based on edit distance. In: SIGMOD, pp. 915–926 (2010)
7. Xiao, C., Wang, W., Lin, X., Yu, J.: Efficient similarity joins for near duplicate detection. In: WWW, pp. 131–140 (2008)
8. Vernica, R., Carey, M., Li, C.: Efficient parallel set-similarity joins using MapReduce. In: SIGMOD, pp. 495–506 (2010)
9. Arasu, A., Ganti, V., Kaushik, R.: Efficient exact set-similarity joins. In: VLDB, pp. 918–929 (2006)
10. White, T.: Hadoop: The Definitive Guide. Yahoo Press (2010)
11. Hernández, M., Stolfo, S.: The merge/purge problem for large databases. In: SIGMOD, p. 138 (1995)
12. Dong, X., Halevy, A., Madhavan, J.: Reference reconciliation in complex information spaces. In: SIGMOD, pp. 85–96 (2005)
13. Elmagarmid, A., Ipeirotis, P., Verykios, V.: Duplicate record detection: A survey. IEEE Transactions on Knowledge and Data Engineering, 1–16 (2007)
14. Naumann, F., Herschel, M.: An Introduction to Duplicate Detection. Synthesis Lectures on Data Management 2(1), 1–87 (2010)
15. Hernández, M., Stolfo, S.: Real-world data is dirty: Data cleansing and the merge/purge problem. Data Mining and Knowledge Discovery 2(1), 9–37 (1998)

16. Broder, A., Glassman, S., Manasse, M., Zweig, G.: Syntactic clustering of the web. Computer Networks and ISDN Systems 29(8-13), 1157–1166 (1997)
17. Henzinger, M.: Finding near-duplicate web pages: a large-scale evaluation of algorithms. In: SIGIR, pp. 284–291 (2006)
18. Cohen, E., Datar, M., Fujiwara, S., Gionis, A., Indyk, P., Motwani, R., Ullman, J., Yang, C.: Finding interesting associations without support pruning. IEEE Transactions on Knowledge and Data Engineering 13(1), 64–78 (2002)
19. Kim, H., Lee, D.: HARRA: fast iterative hashed record linkage for large-scale data collections. In: EDBT, pp. 525–536 (2010)
20. Chaudhuri, S., Ganti, V., Kaushik, R.: A primitive operator for similarity joins in data cleaning. In: ICDE (2006)