International Journal of Computer Applications (0975 – 8887)
Volume 120 – No.18, June 2015

Identification of Similar Strings in a Dataset using Scalable Join

Khalid F. Alfatmi
Department of Computer Engineering,
GES's R. H. Sapat College of Engineering,
Savitribai Phule Pune University, India

Archana S. Vaidya
Professor, Department of Computer Engineering,
GES's R. H. Sapat College of Engineering,
Savitribai Phule Pune University, India

ABSTRACT
Similarity Join plays an important role in data integration and cleansing, record linkage and data de-duplication. It finds similar sting pairs from collections of strings. If two strings are similar they share a common token. Number of approaches has been proposed for in-memory string similarity joins. But due to the rising era of big data, demands for scalable algorithms to support large scale string similarity joins arises. The proposed architecture uses the MapReduce concept and is based on inverted index and multiple prefix filtering methods. The prefix filtering is made of different global ordering which reduces the number of candidate pairs significantly, thus improving the pruning power as compared to other approach.

Keywords
Similarity Join, MapReduce, Big data, Hadoop

1. INTRODUCTION
The string similarity join finds all *similar* string pairs from the given set of strings. String similarity join plays an important role in many real-world applications like data cleansing and integration, and duplicate detection. There are number of applications that require detecting similar pairs of records. List of possible applications includes: detecting near duplicate web-pages in web crawling, document clustering, plagiarism detection, master data management, making recommendations to users based on their similarity to other users in query refinement [1][2]. For example, in master-data-management applications, a system has to identify that names "Rajiv S Kapoor", "Kapoor, Rajiv", and "Rajiv Subodh Kapoor" are potentially referring to the same person.

Detecting such similar pairs is a challenge today, as applications are dealing with vast amounts of data. The size of data is so large that it usually do not fit in the main memory of one machine. The similarity between two strings is usually calculated by similarity functions. The two main types of similarity functions are: set-based similarity function (e.g. Cosine, Jaccard, Dice) and character-based similarity functions (e.g. Edit distance). Number of existing string similarity-join methods used in-memory algorithms which are restricted to a particular size of dataset. But the increased use of big data now poses new challenges for large-scale string similarity joins and demands for new scalable algorithms.

Large number of algorithms proposed for string similarity joins take assistance from inverted index. In this approach, they adopt a two stage filter and refine strategy in identifying similar string pairs- first to generate candidate pair after traversing the inverted index; and next to verify candidate pair by computing similarity. But on other side, most of these algorithms suffer from low pruning power, or they incur too much computation to improve the pruning power. Hence the proposed system is a multiple prefix filtering method based on global ordering.

Problem Definition

Given a collection of strings*'S'* and a threshold value *'Ø'* (theta), the string similarity join *SIM* finds out all the similar string pairs from the collection, such that *SIM(si,sj) ≥ Ø*.

The Similarity between two strings is quantified by the similarity functions/similarity metrics. The output of this similarity function is compared with a predefined threshold value. Mainly two types of similarity metrics are: character based and set-based similarity metrics.

1.1 Character-based similarity function:
This function calculates the similarity between two strings based on character transformations. They are efficient for capturing typographical error. Edit Distance is one of the representative of character based similarity function. *Edit distance* between two strings is nothing but the minimum number of edit operations that transform one string into another [13]. Allowable edit operations are- deletion of characters, replacing a character in the string by another or inserting new character.

For example consider two strings *x*="pratik" and *y*="prateek".

Their edit distance ED(*x*,*y*)= 2, since the first string can be transformed to second by transforming two characters. Two strings are said to be similar w.r.t. the edit distance metric only if their edit distance is not larger than a given threshold 'τ'.

1.2 Set-based similarity function:
The set based similarity function first transform strings into sets of tokens. A token can be either a word or a*n*-gram. In*n*-gram- a string's substrings with length *n* is used to generate the set, where the substring with length *n* is called a*n*-gram. For example, the 2-gram set of "bsnl" is {"bs", "sn", "nl"}. Token-based metrics are found to be suitable for long strings, e.g., documents. The three well-known set-based similarity functions are Jaccard, Dice, and Cosine [5].

$$JAC(x,y) = \left|\frac{x \cap y}{x \cup y}\right| \;....\;(1)$$

Where *x*, *y* are two strings.

Two strings are similar w. r. t. the set-based similarity function if their similarity is not smaller than the given threshold value. The proposed system will use Jaccard function as similarity function and tokens will be the words of strings.

2. LITERATURE REVIEW

The string similarity join problem can be divided into two categories as: In-memory similarity joins and MapReduce based similarity joins. Number of approaches has been proposed for in-memory similarity join which are restricted with the dataset size. Few of the approaches have been done for the MapReduce based string similarity join which still needs to be upgraded.

To avoid verifying every pair of strings in the data set and improve performance, string similarity join operation consists of two phases: candidate generation and verification [2]. In the candidate generation phase, the signature assignment process is invoked. Recent works are typically built on top of some traditional indexing methods as- tree based and inverted index based. In [8], the Trie-tree-based approach was proposed for edit similarity search, where an in-memory Trie-tree is built. This support edit similarity search by incrementally probing the nodes. The edit similarity join method based on the Trie-tree was proposed in [7], in which sub-trie pruning techniques are applied. In [11], a B+-tree based method was proposed to support edit similarity queries. This method transforms the strings into digits and indexes them in the B+-tree. However, these algorithms are limited to in-memory processing, and so are not efficient and scalable for processing large scale data set.

The inverted index based methods are of the fact that similar strings share common parts, and hence they transform the similarity constraints into set overlap constraints. To prune false positives, the PPJoin method applies the position information of the prefix tokens of the string. Based on the PPJoin, the PPJoin+ uses the position information of suffix tokens to prune false positives further [9]. As these methods need to merge the inverted lists during the candidate generation phase, further some optimization techniques for the inverted list merging were introduced. The exact computation method proposed in [1] is based on the pigeon hole principle which transforms similarity constraints into Hamming distance constraints and transforms each record into a binary vector. This binary vector is divided into partitions and then hashed into signatures. The strings that produce the same signatures are considered as candidate pairs. However, the signature scheme is very time-consuming and introduces unnecessary false positives.

In [2], Vernica proposed a prefix filtering based method, which used a filter-and-verification framework. In the filter step, they selected some tokens from each string and generated a set of candidate pairs who share a common token. In the verification step, they verified the candidate pairs to generate the final answers. One of the big limitations of this approach is low pruning power. A single token results to be very short and usually has low selectivity, hence many dissimilar pairs will share a same token and cannot be pruned.

3. PROPOSED WORK

MapReduce is a data intensive programming model which works on a cluster of nodes. It consists of two main functions; map and reduce. The map function read in the input data, and emits multiple intermediate <key, value> pairs. Then the Reduce functions merge these <key, value> pairs in such a way that all values associated with the same key are paired together. The map & reduce programs are automatically parallelized and scheduled on a large cluster of machines. Further the process of scheduling the jobs, balancing maintenance on different nodes, detection of errors and recovery from those errors are automatically managed by the computing platform such as Hadoop. Hence, the users only need to concern with the implementation detail of the algorithm. This helps users without parallel programming experience to easily utilize the computing resources.

3.1 Single Global ordering

An inverted index based string similarity joins algorithm adopts three-step approach in identifying similar string pairs. This approach is known as filter and refines approach, which consists of following steps:

i) To generate an inverted index from the given prefix tokens.

ii) To generate candidate pairs by traversing through the inverted index built in step1.

iii) To verify the candidate pairs by computing the similarities between them.

3.2 Multiple Global ordering

By applying the single global ordering method, the candidate pairs can be reduced significantly as compared to simple approach, where every two strings that share common tokens are considered as a candidate pair. But, in case of large-scale data sets, the numbers of candidate pair results to be very large and needs to be reduced considerably. Therefore, multiple global ordering can be used to reduce the number of candidate pairs. One of the approach for implementation can be, to repeat the use of single global ordering for each ordering. Then derive the overlap among these sets of candidate pairs. However, this process for candidate pair generation is costly. Hence, this problem is taken care by the proposed system.

Proposed system is based on multiple prefix filtering technique, which applies different global orderings in a pipelined manner. In Figure 1, a set of global ordering is applied on different stages where O_1 is selected as a basis to build inverted index for prefix tokens. The candidate pairs are generated for each string based on its prefixes. A pipelining process is used to prune the false positives in advance. The candidate pairs are continuously checked in pipelining order which significantly reduces their size.

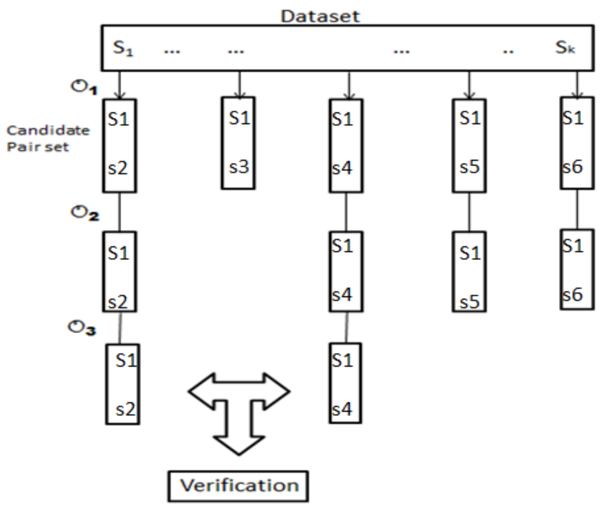

Fig 1: Pipeline of Global Ordering

MATHEMATICAL MODEL

P={ S, O_g, k, F, ø , W}

Where
S: is set of strings with different length, each string made of words,
O_g: is set of global ordering
K: No. of prefix filtering
F: set of map and reduce functions
Ø: threshold value (between 0 and 1)
W: output pair of similar string

Consider if
S={s1,s2,s3,s4}
K=3
O_g: {O1,O2,O3}
Ø : 0.8
F : {F1, F2, F3}
F1 : generates token universe and term frequency order
F2 : canonicalize dataset according to designated global ordering and balance dataset on all nodes
F3: implement the MapReduce based String Similarity join algorithm

Then, if srings s1 and s2 are similar, the output will be, W={s1,s2}

The algorithm for String Similarity Join using Global Ordering is outlined as follows.

Algorithm 1:

Input: *S=input dataset; Ø=threshold value; k= number of prefix filtering*

Output: *All pair of strings with similarity SIM(si,sj)≥ Ø*

1. Tokenize each string in the dataset 'S'.
2. Sort them based on global ordering 'O' based on term frequency TF.
3. Other prefix tokens are derived during canonicalization process.
4. Sorted strings are processed sequentially in two phases- Candidate Generation and Verification.
5. For each prefix token, its inverted index will be scanned using length filtering method and then multiple prefix filtering.
6. String pairs satisfying all filter conditions are considered as candidate pairs
7. To avoid duplicate pair verification, candidates are counted for number of occurrences.
8. Finally pairwise verification is done with respect to given threshold value.

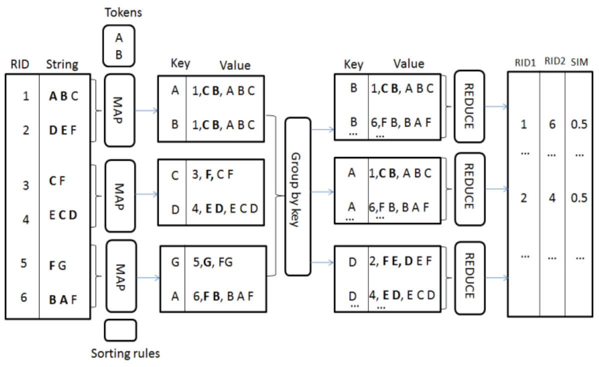

Fig 2: MapReduce based String Similarity Join

To apply the above algorithm to the MapReduce framework, the main challenge is to assign keys to candidate string pairs. A candidate pair of strings must have one common prefix token under any global ordering, prefix tokens of a string are choose as its keys and the string content as the value so that strings with the same prefix token can be shuffled to the same reducer.

Following is the proposed MapReduce based String Similarity Join Algorithm

Algorithm 2:

1. Generate the token universe U and term frequency order <*sorted token file*>
2. Load the token files and sorting rules on each datanode.
3. In Map, prefix token set is derived for each string.
4. The new value from Map will be set of record id, prefix token and original record
5. String pairs with the same key will be shuffled to same node.
6. In reduce, string with same key will be verified with respect to given threshold value using Jaccard function.
7. To avoid duplicate pair verification, candidates are counted for number of occurrences.
8. Pair of similar string with its threshold value will be given as output

4. RESULTS

Proposed work implements similarity join with MapReduce based on inverted index and multiple prefix filtering methods. It will also try to reduce memory overheads, communication between work nodes by distributing appropriate proportion of key value pairs. It is compared with implementation of parallel set-similarity joins [2] which is based on signature similarity join. Implementation of both algorithms is tested on DBLP dataset.

DBLP is a computer science bibliography snapshot downloaded from DBLP website[12]. It contains records which are concatenation of author name and title of publication.

The performance of proposed system is evaluated on Hadoop platform on a centralized machine as well as on a cluster of 4 computer nodes, among which one is configured as the master and the others are configured as slaves.

Centralized system: Experiment performed on single machine with core i7 processor*2.0*GHz x 2 and 8GB of main memory. The operating system is Ubuntu 14.04 and all algorithms are implemented in JDK 1.7.

Taking record size=100 and threshold value =0.8 we get the following results.

Table 1: Comparative Result on Centralized system

Different Stages of Join Operation	Time taken by Proposed Approach	Time taken by Self Join
Record Build	11.45sec	11.35sec
Record Balance	17.54sec	18.45sec
Join operation	**105.38sec**	**110.27sec**
No. of matching records	14	14

Distributed and Parallel System:

All nodes have the same configurations as follows:

- CPU: Intel core 2 Duo;

- Main memory: 2 GB;

- Disk: 320 GB;

- OS: Ubuntu 14.0;

- Software environment: Hadoop-2.6.0 and JDK-1.7.0.

Taking 31,7004 records and the join threshold is set to 0.6.we get following results.

Table 2: Comparative result on Distributed System

Stages of Join operation	Time taken by Proposed Approach
Record Build	33
Record Balance	30
Join Time	**95.54**
No. of matching records	15267

5. CONCLUSION
The proposed approach is a MapReduce-based framework for scalable string similarity joins. The system generates signatures for strings, which acts as keys and the stings itself acts as values for the MapReduce framework. This method is based on multiple prefix filters which applies different global ordering to reduce the number of candidate pairs. It is executed on centralized system as well as distributed computing environment. When compared to the parallel set similarity join approach, the proposed approach is much more efficient and scalable.

6. FUTURE SCOPE
Further work can be carried on to find an efficient mechanism for generating total number of global ordering needed as well as their sequence of application, to reduce the candidate pair size.

7. ACKNOWLEDGEMENT
I would like to express my sentiments of gratitude to all who rendered their valuable guidance for this work. I would like to thank Dr. P. C. Kulkarni, Principal, GES's. R. H. Sapat College of Engineering Nashik, for providing me strong platform to develop my skills and capabilities.

I am also thankful to Prof. N. V. Alone, Head of Department, Computer Engineering and my guide Prof. A. S. Vaidya for helping and guiding me with the topic and also providing me with adequate facilities, ways and means by which I was able to complete this paper.

8. REFERENCES
[1] C. Rong, Wei Lu, Xiaoli Wang, Xiaoyong Du and Anthony K.H. Tung, "Efficient and Scalabe Processing of String Similarity Join," IEEE Transactions on Knowledge and Data Engineering, VOL. 25, 2013, pages 2217-2230.

[2] R. Vernica, M. J. Carey, and C. Li, "Efficient Parallel Set Similarity Joins using MapReduce," In SIGMOD, 2010, pages 495-502.

[3] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," In OSDI, pages 137- 150, 2014.

[4] NikolausAugsten, Michael H Bohlen, "Similarity Joins in Relational Database Systems," Morgan & Claypool publishers.

[5] Younghoon Kim, Kyuseok Shim, "Parallel Top-K Similarity Join Algorithms using MapReduce," IEEE 28th International Conference on Data Engineering, 2012.

[6] A. Arasu, V. Ganti, and R. Kaushik, "Efficient Exact Set-Similarity Joins," Proc. 32nd International Conf. Very Large Data Bases, pp. 918-929, 2009.

[7] Yu Jiang, Guoliang Li, JinhuaFeng, Wen-Syan Li, "String Similarity Joins: An Experimental evaluation", International Conference on Very LargeDataBases, Vol.7, No.8., 2014.

[8] Wang, J. Feng, and G. Li, "Trie-Join: Efficient Trie-Based String Similarity Joins with Edit-Distance Constraints," Proc. VLDB Endowment, vol. 3, nos. 1/2, pp. 1219-1230, 2010.

[9] C. Xiao, W. Wang, X. Lin, and J. Yu, "Efficient Similarity Joins for Near Duplicate Detection," Proc. International Conf. World Wide Web, pp. 131-140, 2008.

[10] A. Elmagarmid, P. Ipeirotis, and V. Verykios, "Duplicate Record Detection: A Survey," IEEE Trans. Knowledge and Data Eng., vol. 19, no. 1, pp. 1-16, Jan. 2007.

[11] Z. Zhang, M. Hadjieleftheriou, B. Ooi, and D.Srivastava, "Bed-Tree: An All-Purpose Index Structure for String Similarity Search Based on Edit Distance," Proc. ACM SIGMOD International Conf. Management of Data, pp. 915-926, 2010.

[12] Dataset https://www.informatik.uni-trier.de/~ley/db

[13] D. Deng, G. Li, S. Hao,"MassJoin: A MapReduce-based Method for Scalable String Similarity Joins", IEEE 30th International Conference on Data Engineering, pp. 340-351, 2014.