ISSN:0975-9646

Survey of Scalable String Similarity Joins

Khalid F. Alfatmi[1], Archana S. Vaidya[2]

Department of Computer Engineering,
Savitribai Phule Pune University,
Maharashtra, India

***Abstract*— Similarity Join is an important operation in data integration and cleansing, record linkage, data deduplication and pattern matching. It finds similar sting pairs from two collections of strings. Number of approaches have been proposed as well as compared for string similarity joins. The rising era of big data demands for scalable algorithms to support large scale string similarity joins. In this paper we study the string similarity joins, their use. Further we look at three different techniques for scalable string similarity join using MapReduce, which are- Parallel set-similarity join, MGJoin and MassJoin. Finally, we try to compare them based on some common characteristics.**

***Keywords*— Similarity Join, MapReduce**

I. Introduction

A string similarity join between two sets of strings finds all *similar* string pairs from the two sets. String similarity join can play an important role in many real-world applications, e.g., data cleansing and integration, and duplicate detection. Given two collections of strings, e.g., products and city names, the string similarity join problem is to find all similar string pairs from the two collections of strings. The similarity between two strings is usually quantified by similarity functions. There are two main types of similarity functions: set-based similarity functions (e.g., Jaccard, Cosine, Dice) and character-based similarity functions (e.g., Edit distance), which we will discuss in the section II.

Number of existing similarity-join methods used in-memory algorithms which are restricted to a particular size of dataset. But the rise of big data now poses new challenges for large-scale string similarity joins and demands for new scalable algorithms. In this paper we study the three different scalable string similarity joins. The first approach in this direction is by Vernica[1], where they propose a three stage method for end-to-end set-similarity joins. The first stage generates appropriate signatures for the data. In the second stage record IDs and join attribute values are extracted from each record and then they are distributed across the reducers such that pair sharing signature go to a common reducer. The reducer outputs RID pairs of similar records. Finally the third stage generates actual pairs of joined records.

However the Parallel set similarity join considers only one token at a time which increases the candidate keys and thereby reduces the pruning power. Hence a new MapReduce based framework came into picture, which extends to support set-based similarity functions as well as character based similarity function- called MassJoin. This technique uses the *Filter* and *Verification* step to produce similar string pair. It also uses *light weight filter* to decrease the number of candidate pairs which reduces transmission and computation cost. The MassJoin framework also overcomes the low pruning power and skewed problem faced in prefix filtering method [3]. Both this methods fall under the filter and refine category.

The MGJoin proposes a multiple prefix filtering method in which different global ordering are applied on the data such that the number of candidate pairs are reduced significantly [2]. The basic aim behind the MGJoin and MassJoin method is to reduce the number of candidate keys with reasonable computing cost which significantly improves the filtering power.

II. Priliminary

Given two collections of strings the string similarity join finds out all the similar pairs from the two collections. Whether the two strings are similar, is quantified by the similarity functions/similarity metrics. The output of the function is compared with a predefined threshold value. The two main types of similarity metrics are: character-based similarity metrics and set-based similarity metrics.

Character-based similarity function: This function quantifies the similarity between two strings based on character transformations. They are fit for capturing typographical error. Edit Distance is a representative of character based similarity function. *Edit distance* between two strings is the minimum number of edit operations that transform one string into another [6][7]. Allowable edit operations are- insertion as well as deletion of characters, and replacing a character in the string by another character. For example consider two strings p=”jhon” and q=”john”. Their edit distance ED(p,q)= 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 if their edit distance is not larger than a given threshold ‘τ’.

Set-based similarity function: This function first transform strings into sets of tokens. A token can be either a word or a n-gram. The n-gram uses a string’s substrings with length n to generate the set, where the substring with length n is called a n-gram. For example, the 2-gram set of “imdb” is {“im”, “md”, “db”}. The token-based metrics are suitable for long strings, e.g., documents. The three well-known set-based similarity functions are Jaccard, Cosine, and Dice[5].

$$JAC(r,s) = \frac{|r \cap s|}{|r \cup s|} \qquad COS(r,s) = \frac{|r \cap s|}{\sqrt{|r|.|s|}}$$

$$DICE(r,s) = \frac{2|r \cap s|}{|r| + |s|}$$

where *r, s* are two strings.
Two strings are said to be similar w.r.t. the set-based similarity function if their similarity is not smaller than a threshold '*δ'*.

MapReduce: MapReduce is a programming model proposed by Google. It is used for processing and generating large datasets [4]. The actual computations are specified by the user in terms of two separate functions as map and reduce. These computations are automatically parallelized across large-scale clusters of machines by the underlying runtime system. The computation takes a set of key/value pairs as input, and produces a set of key/value pairs as output. The MapReduce computation takes place as two functions: *map* and *reduce*. The *Map function* which takes an input pair, produces a set of *intermediate* key/value pairs. The MapReduce library is responsible for grouping together all intermediate values associated with the same intermediate key and passes them to the reduce function. The *reduce function*, which accepts an intermediate key and a set of values for that key, acts as a reducer by merging the values together to form a smaller set of values. Typically, the reduce function produces just zero or one output value.

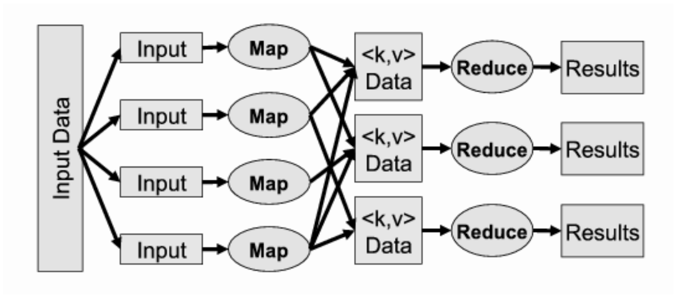

Fig. 1 Dataflow in a MapReduce computation

III. Parallel Set Similarity Framework

The parallel set similarity join is the first approach towards scalable string similarity join. This approach consists of 3 stages as follows[1]:

1. The first stage scans the data and computes frequency of each token and then sorts the token based on frequency; this is called as token ordering.
2. The second stage produces list of similar RID pairs using the prefix filtering principle. Further based on prefix token, the MapReduce framework groups the RID and join-attribute value pairs.
3. Finally the third stage generates pair of similar records by using list of similar RID pairs and the original data.

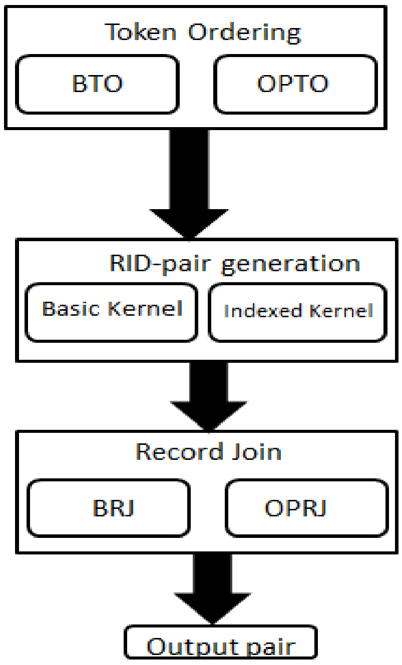

Fig. 2 Parallel set similarity join Framework

Fig. 2 shows the functional block diagram of Parallel set similarity join.

The 3 stages based on self-join can discussed further in detail:

A. Token Ordering

There are two methods of token ordering in the first stage-

1. Basic Token Ordering (BTO)- relies on two phases of MapReduce in which first phase computes the frequency of each token and the second phase sorts the token based on their frequencies.
2. One Phase Token Ordering (OPTO)- in this approach the list of token is explicitly sorted in memory. Hence it uses only one MapReduce phase.

B. RID-Pair generation

This stage also called as kernel, scans the original data and extracts prefix of each record using the token order computed by the first stage. There are two approaches of finding the RID pair of similar records:

1. Basic Kernel (BK)- In this function the reducer uses a nested loop approach to compute similarity of join attribute values. The map function extracts RID and join attribute values after retrieving the original data. It tokenizes the join attribute and computes the prefix length. At the last it uses individual or group token routing strategy to generate output pair.
2. Indexed Kernel (PK)- This function uses existing set similarity join algorithm- PPJoin+ to find RID pairs of similar records. Hence it is called as PPJoin+ Kernel(PK).

C. **Record Join**

The final stage is used to join the records of RID pairs of, generated in stage 2.

1. Basic Record Join(BRJ)- uses two MapReduce phases. The first phase fills in the record information for each half of each pair and the remaining half is taken care in the second phase.
2. One Phase Record Join(OPRJ)- uses only one MapReduce phase where the RID pairs are broadcast and loaded at map function before the function consumes the input data.

IV. MGJoin Framework

Many algorithms proposed for string similarity join take assistance from inverted index. They adopt a two stage filter and refine strategy in identifying similar string pairs: 1. To generate candidate pair after traversing the inverted index; and 2. To verify candidate pair by computing similarity. But in general most of these algorithms suffer from low pruning power, or they incur too much computation to improve the pruning power. Hence a multiple prefix filtering method based on global ordering is proposed called as MGJoin.

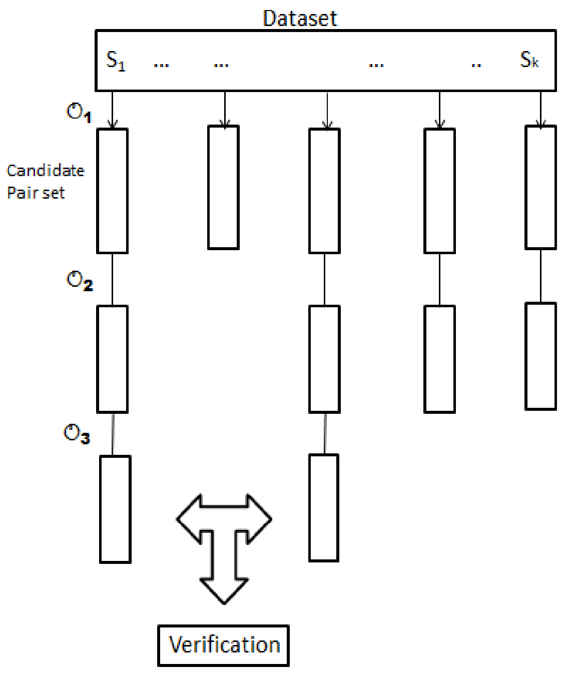

Fig. 3 MGJoin- Multiple prefix filtering

MGJoin is based on multiple prefix filtering technique. It applies different global orderings in a pipelined manner. In Fig. 3, a set of global ordering is applied on different stages where $\mathcal{O}_1$ is selected as a basis to build inverted index for prefix tokens. Further candidate pairs are generated for each string based on its prefixes. A pipelining processing is used to prune the false positives in advance. The candidate pairs are continuously checked in pipelining order, significantly reducing their size.

V. MassJoin Framework

MassJoin is a MapReduce-based string similarity join algorithm, which can support both set-based similarity functions and character-based similarity functions. Consider there are two set of strings; *R* and *S* both containing multiple strings. <*sid*> and <*rid*> are the ids given to the strings in the set *S* and *R* respectively. The working of MassJoin shown in fig. 4, can be deduced in following steps:

A. Signature Generation

The character based similarity function depends on given edit-distance threshold and generates a fixed number of signatures. But, for a set-based similarity functions, the number of signatures depends on the string lengths. Hence in MassJoin a new algorithm is used to generate signatures. Basically two methods are used for signature generation- Position-aware method and Muti-match-aware method. The two methods can also be used simultaneously known as hybrid method. These methods will reduce the number of signatures generated simultaneously avoiding false-negatives.

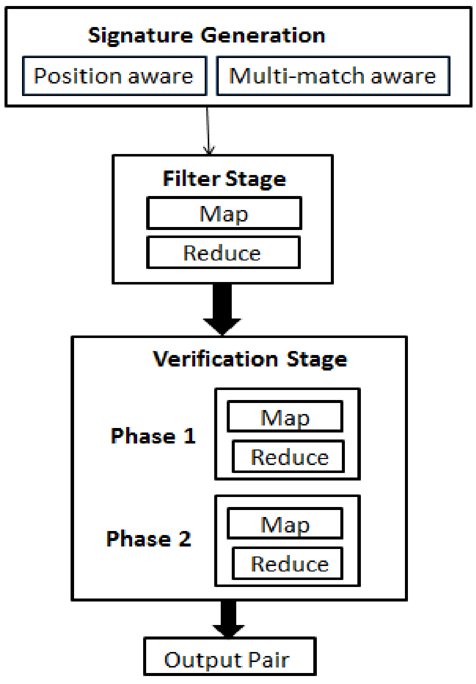

Fig. 4 MassJoin Framework

B. Filter Stage

This stage generates candidate pairs using techniques mentioned in Signature generation stage. The map phase uses the signatures as keys and string as values. As two similar strings share a same key, they are shuffled to the same reduce task. String Ids are used in place of strings which reduces the transmission cost. Each reduce node in the reduce phase, takes a key value pair as input, which consist of signature and the list of strings containing the signature. Next it splits the list into two groups i.e separate for <*sid*> and <*rid*>.

C. Verification Stage

This stage verifies the candidate pairs generated from the filter stage. It provides with a two-phase method to handle two important goals; first- to eliminate duplicates which arise due to two string sharing multiple signature and second- to replace the id in candidate pair with real string.

TABLE 1
COMPARISONOF PARALLEL SET SIMILARITY JOIN, MGJOIN AND MASSJOIN

Comparison Points	**Parallel Set-Similarity Join**	**MGJoin**	**MassJoin**
Basic Technique	3 stages- token allotment, pairing and record join	Multiple global ordering	Filter and Verification
Similarity function supported	Set based similarity function	Set based similarity function	Character based as well as Set based similarity function
Joins Supported	Self, RS Join		Self-Join
Number of tokens	Single	Pipelined	Multiple
Performance	First approach, hence low pruning power, skewed problem	Outperforms PPJoin+ and other state of the art methods	High pruning power with light weight filters

VI. Proposed System

In the proposed system, partition based approach based on prefix filtering will be implemented which will support both, the set based similarity function and character based similarity function. In this technique, signatures will be generated with the hybrid approach depending on the similarity function. The filter and verification phase will have only one map and reduce function in which the pruning will be done based on global ordering. The filters can be used in both the map and reduce phase to minimize the number of candidate pair. This will enhance the pruning power as well as improve the performance of the overall system.

VII. Conclusions

This paper provides a comprehensive survey of existing scalable string similarity join algorithms, including MGJoin, MassJoin and parallel set similarity join. The parallel set similarity join is a three stage based method. It considers a single token as key which leads to low pruning power and skewed problem. Whereas MassJoin take care of the shortcomings faced by previous two approaches efficiently. It implements *character based* as well as *set based* similarity function, suitable for short strings as well as large documents. It also implements the merging technique and *light weight filters,* that improves the performance of MassJoin significantly over MGJoin and parallel set-similarity joins.

Acknowledgment

We are glad to express our sentiments of gratitude to all who rendered their valuable guidance to us for this work. We would like to thank Prof. Dr. P. C. Kulkarni, Principal, G. E. S. R. H. Sapat College of Engg., Nashik. We are also thankful to Prof. N. V. Alone, Head of Department, Computer Engg., G. E. S. R. H. Sapat College of Engg., Nashik. We thank the researchers for publishing their research work as a guideline.

References

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

[2] 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.

[3] D. Deng, G. Li, S. Hao, Wang and J.Feng, “MassJoin: A MapReduce-based Method for Scalabe String Similarity Joins,” ICDE Conference, 2014.

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

[5] NikolausAugsten, Michael H Bohlen, “Similarity Joins in Relational Database Systems,” Morgan & Claypool publishers.

[6] G. Li, D. Deng, J. Wang, and J. Feng, “Pass-join: A partition-based method for similarity joins” *PVLDB*, 5(3):253-264, 2011.

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

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