Survey of String Similarity Join Algorithms on Large Scale Data

P.Selvaramalakshmi
Research Scholar
Dept. of Computer Science
Bishop Heber College (Autonomous) Tiruchirappalli, Tamilnadu, India.

Dr. S. Hari Ganesh
Asst. Professor
Dept of Computer Science
H.H. The Rajah's College Puddukottai Tamilnadu, India.

J.James Manoharan
Asst. Professor
Dept. of Computer Applications
Bishop Heber College (Autonomous) Tiruchirappalli, Tamilnadu, India.

Absract: String Similarity Joins plays a very important role in many applications related to data integration, data cleaning, and pattern matching and data reduplication. A String Similarity Join finds pair of items from two data sets such that the pairs similarity value is no less than a given threshold value. Several approaches have been proposed and compared for string similarity join. The past two decades there are many algorithms have been projected for string similarity join. But the presented algorithms have not been methodically compared under the same framework. Many algorithms where tested only on particular datasets. This makes difficult for the practitioners to make a decision on which algorithms should be used at different situation. In this paper we study the string similarity joins algorithms and their usability. We approached three different techniques such as Parallel Set, MGJoin and MassJoin for scalable string similarity join. Finally we compare these algorithm based on some general characteristics.
.

Keywords- Data Integration, Data cleaning, String Similarity join, Pattern Matching.

I. INTRODUCTION

A string similarity join among two sets of strings find every *similar* string pairs from the two set of data. The association of two strings can be quantified by similarity metrics between two strings [1]. String similarity join plays a very important role in several real-world applications such as duplication detection, data cleaning and integration [5]. Given two set of strings, for example products and city names, the string similarity join trouble to find out all similar string pairs from the two collections.

The brute-force algorithm that enumerates each string pair and ensures whether the two strings in the pair are similar is very expensive. Many algorithms have been proposed in the past two decades to solve this problem, In the past two decades data mining has emerge as a major research area .This is main due to the inter-disciplinary character of the subject and the various series of application domain in which data mining based products and technique are being employed. This includes medicine, education and marketing bioinformatics, clinical research, genetics and research.

II. PRELIMINARIES

Given two set of strings, the string similarity join problem is to find all the similar pairs from the two sets. Similarity metric is the calculation that can be used to check whether two strings are similar or not [9]. The existing similarity metrics can be categorized into character based similarity metrics and token-based similarity metrics.

Character-based Metrics: These metrics measure the similarity between two strings based on character transformations. One depiction of character-based metric is edit distance. It finds the distance between two strings at the least number of edit operations needed to change one string in to other string, where the permitted edit operations such as insertion, deletion, and substitution [2][3]. For example, consider two strings *"xnrs"* and "*pxnrs"*. The edit distance of *("xnrs","pxnrs")* = 1, since the first one can be transformed to the second one by inserting a character *"p".*

Token-based Metrics: These metrics first alter strings into sets of tokens and then use the set-based similarity metrics to measure the similarity. The token-based metrics are appropriate for long strings. Two strategies are generally used to change strings into sets: (1) tokenization and (2) q-grams. The first one tokenizes strings based on particular characters. The second one uses strings substrings with length q to produce the set, where the substring with length q is called a q-gram.

Map Reduce: MapReduce is one of the famous frameworks projected by Google to make possible processing of large-scale data in parallel. The actual computations are specified by the user in terms of two separate functions as map and reduced [7]. These computations are automatically parallelized transversely large-scale clusters of machines by the original runtime system. In MapReduce, data is initially partitioned across the nodes of a cluster and stored in a distributed file system (DFS). Data is represented as (value,key) pairs. The calculation is articulated using two functions:

Map (k1, v1) → *list (k2, v2);*

Reduce (k2, list (v2)) → *list (k3, v3);*

The map function takes as input a (key, value) pair, denoted by (k1, v1), and produces as output a list of new (key, value) pairs, denoted by list(k2, v2).The reduce function takes as the input one of the keys output from the map (k2) and a list containing all the values output with that key (list(v2)). In return, the reduce function outputs a new list of (key, value) pairs, expressed by list (k3, v3).

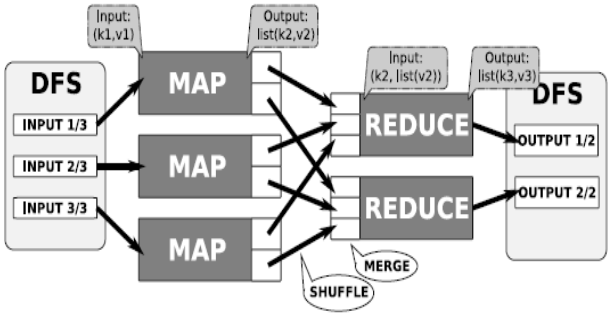

Figure 1 Dataflow in a MapReduce calculation

Figure 1 depicts the Flow of data in a MapReduce Calculation.

III. PARALLEL SET SIMILARITY FRAMEWORK

The parallel set similarity join is the primary approach towards scalable string similarity join. This method consists of 3 stages as follows:

1. In the first stage the data can be scanned and then frequency of each token can be calculated. Finally the tokens are sorted based on frequency. This is called as token ordering.
2. In the second stage list of similar RID pairs produced by using the prefix filtering standard. Further the MapReduce framework groups the RID and join attribute value pairs based on prefix token,
3. In the third stage, by using list of similar RID pairs and the original data the pair of similar records are generated.

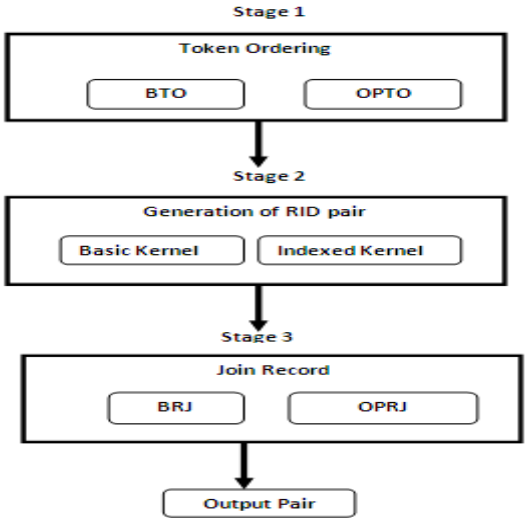

Figure 2 Framework of Parallel set Similarity Join

Figure 2 depicts the functional block diagram of Parallel set Similarity Join. The three stages based on self-join can discussed further in detail:

Stage 1: *Token Ordering*

We consider two possible alternate methods for ordering the tokens in the first stage. Both methods take as input the original records and produce a list of the tokens that appear in their join-attributes ordered increasingly by their global frequency of usage.

A.Basic Token Ordering (BTO):

Basic Token Ordering ("BTO") depends on two MapReduce phases. The frequency of each token is computed in the first phase then in the second phase the tokens are arranged based on their frequencies. In the first phase, the map function receives as input the original records. For each record, the function extracts the value of the join attribute and tokenizes it. The second phase uses MapReduce to arrange the pairs of tokens and frequencies from the first phase [4]. The Map function swap the input keys and values so that the input pairs of the reduce function are arranged based on their frequencies. This phase uses accurately one reducer so that the result is a completely ordered list of tokens.

B.One-Phase Token Ordering (OPTO):

The list of tokens can be much smaller than the original data size in OPTO [10]. We can sort the tokens explicitly in memory instead of MapReduce, The reduce function in One-Phase Token Ordering gets as input a list of tokens and their limited counts. For each token, the function computes its whole count and stores the information.

Stage 2: *RID-Pair Generation*

The second stage of the join, called the "*Kernel*", which scans the original input data and extracts the prefix of each record using the token order computed by the first stage. In general the list of distinct tokens is much smaller and grows much more slowly than the list of records. We thus assume that the set of tokens fits in memory. Based on the prefix tokens, we extract the RID and the join-attribute value of each one record, and then distribute these record projections to reducers. The join-attribute values that share at least one prefix token are verified at a reducer.

Approaches to find out the RID pair of similar records:

A. Basic Kernel (BK): Computation of similarity in join attribute values can be calculated by nested loop approach. The map function fetches RID and join attribute values after taking out the original data , then this function can tokenizes the join attribute and calculates the prefix length. Finally it uses individual or group token routing strategy to produce output pair.

B. Indexed Kernel (PK): This function uses already availability set of similarity join algorithm like *PPJoin+* to find RID pairs of similar records. Hence it is known as PPJoin+ Kernel (PK).

Stage 3: *Record Join*

In the final stage of the algorithm is to use the RID pairs generated in the second stage to join their records and there are two approaches used in this stage. The main idea is to first fill in the record information for each half of the pair and then use the two halves to build the complete record pair. The two approaches differ in the way the list of RID pairs is presented as input. In the first approach, called Basic Record Join ("*BRJ*"), the list of RID pairs is treated as a normal MapReduce input [6], is presented as input to the map functions. In the second approach, called One-Phase Record Join ("*OPRJ*"), the list is broadcast to all the maps and loaded before reading the input data. Duplicate RID pairs from the previous stage are eliminated in this stage.

IV. MGJOIN FRAMEWORK

Several algorithms were projected for string similarity join. They adopt a two stage filter and refine strategy in identifying similar string pairs:

1. Candidate pair can be generated after traversing the inverted index.
2. The candidate pair can be verified by calculating the similarity.

Most of the algorithms suffer from few pruning power, or they acquire too much calculations are needed to improve the pruning power. Hence a multiple prefix filtering method based on global ordering is projected called as MGJoin. MGJoin is based on multiple prefix filtering technique [10]. It applies various global orderings in a pipelined manner.

V. MASSJOIN FRAMEWORK

MASSJOIN is a scalable MapReduce-based string similarity join algorithm. This algorithm can support both set-based similarity functions and character-based similarity functions [7]. MapReduce contains two most important stages: the filter stage and the verification stage. MassJoin algorithms working principles involves the following steps:

A. Signature Generation

The character based similarity function depends on given edit-distance threshold and generates a fixed number of signatures. Therefore in set-based similarity functions, the number of signatures depends on the string lengths [8]. The signatures can be generated in MassJoin algorithm. MassJoin algorithm has two methods for signature generation one is Position-aware method and another one is Muti-match-aware method [8]. These two methods can be used simultaneously called as hybrid method. This approach will decrease the number of signatures generated simultaneously avoiding false negatives.

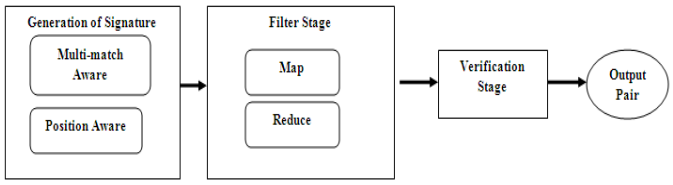

Figure 3 MassJoin Framework

A. Filter Stage

In this filter phase the candidate pairs can be generated by using the filter techniques .The two string r and s must share a signature if two strings r and s are similar. We use the signatures as keys and the strings as values in the Map Phase. The two similar strings must share a same key, that must be shuffled to the similar reduce task. To reduce the transmission cost a key-value pair taken as input.

B. Verification Stage

In the verification stage, the candidate pairs generated from the filter stage can be verified. Candidate pairs generated from the filter stage can be verified in this stage. It provides with a two-phase method to handle two important goals; first to eliminate duplicates which arise due to two strings sharing multiple signature and second to replace the id in candidate pair with real string.

TABLE 1 COMPARITVE ANALYSIS OF PARALLEL SET SIMILARITY JOIN, MGJOIN AND MASSJOIN

String Similarity Join Algorithms	Basic Technique	Supporting Similarity Function	Join Supported	No. of Tokens	Performance

1. Parallel Set-Similarity Join	**Three stages-token allotment , Pairing and Record join**	**Set based similarity function**	**Self Join and RS Join**	**Single Token**	**First approach, Low pruning power, Skewed problem**
2. MGJoin	**Multiple global ordering**	**Set based similarity function**	**Multiple Join**	**Pipelined Tokens**	**Outperforms PPJoin+ other state of the art methods**
3. MassJoin	**Filter and Verification**	**Character Based and Set based**	**Self-Join**	**Multiple**	**High pruning power and Light weight filters**

VII. CONCLUSIONS

This paper provides a broad survey of existing string similarity join algorithms, including MassJoin, MGJoin and Parallel set similarity join. The Parallel set similarity join is a three phase based method. It considers a single token as key which leads to low pruning power and skewed problem. The MassJoin algorithm looks out the shortcoming faced by preceding two approaches efficiently. It implements *character based* in addition to *set based* similarity function which is 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.

REFERENCES

[1] Jiaheng Lu , Chunbin Lin , Wei Wang , Chen Li and Haiyong Wang, "*String Similarity Measures and Joins with Synonyms*", SIGMOD'13, June 22–27, 2013, New York, New York, USA.Copyright 2013 ACM 978-1-4503-2037-5/13/06.

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

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

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

[5] 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.

[6] Wei Wang,Jianbin Qin, Chuan Xiao,Xuemin Lin and Heng Tao Shen, "*VChunkJoin: An Efficient Algorithm for Edit Similarity Joins*", JOURNAL OF LATEX CLASS FILES, VOL. 6, NO. 1, FEBRUARY 2011

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

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

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

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