Al-Badarneh et al., 2016

Volume 2 Issue 1, pp. 214-234

Date of Publication: 19th December, 2016

DOI- https://dx.doi.org/10.20319/mijst.2016.s21.214234

This paper can be cited as: Al-Badarneh, A., Al-Abdi, A., Al-Shboul, S., & Najadat, H. (2016). Survey of Similarity Join Algorithms Based on Mapreduce. MATTER: International Journal of Science and Technology, 2(1), 214-234.

SURVEY OF SIMILARITY JOIN ALGORITHMS BASED ON MAPREDUCE

Amer Al-Badarneh
Computer Information System Department, Jordan University of Science and Technology, Irbid, Jordan
amerb@just.edu.jo

Amnah Al-Abdi
Computer Science Department, Jordan University of Science and Technology, Irbid, Jordan
amalabdi15@cit.just.edu.jo

Sana'a Al-Shboul
Computer Science Department, Jordan University of Science and Technology, Irbid, Jordan
smalshboul15@cit.just.edu.jo

Hassan Najadat
Computer Information System Department, Jordan University of Science and Technology, Irbid, Jordan
najadat@just.edu.jo

Abstract

Similarity Join is a data processing and analysis operation that retrieves all data pairs whose their distance is less than a pre-defined threshold. The similarity join algorithms are used in different real world applications such as finding similarity in documents, images, and strings. In this survey we will explain some of the similarity join algorithms which are based on MapReduce approach. These algorithms are: Set-Similarity Join, SSJ-2R, MRSimJoin, Pair-wise similarity, multi-sig-er method, Trie-join, and PreJoin algorithm. We then make a comparison between these algorithms according to some criteria and discuss the results.

Keywords

Hadoop, MapReduce, Similarity Join

1. Introduction

MapReduce is the main framework for distributed processing, it is providing the requirements to process huge amount of data in a distributed approach. The similarity join algorithms are the algorithms which are used to find the similarity between two objects who's their distance is less than a predefined threshold.

MapReduce approach is an efficient way to solve the similarity join problem because it processes huge datasets by splitting them into distinct chunks which are processed in parallel to reduce the time consuming. There are three functions which are used to find the similarity between two objects depending on the type of the objects: Jaccard similarity to find the similarity between a finite numbers of sets, Cosine similarity to find the similarity between two vectors, and Edit distance (ED) to detect the similarity between two strings. In this survey we will explain some of the similarity join algorithms that are based on MapReduce approach. Theses algorithms will be explained in details in section 2 but we can summarize them just a bit in this section.

SSJ-2R [1] algorithm is an algorithm that is looking for discovering every pair of objects with high similarity larger than the threshold according to some similarity function. This algorithm will be explained in section 2.1.

PreJoin Algorithm [2] is an algorithm that is solving the problem resulting from applying Trie-based similarity join. This algorithm will be explained in section 2.2

Pair-wise similarity computation [3] is assigning objects to multiple overlapping clusters, but redundant pair's will be produced because similar objects may share more than one cluster, in section 2.3 we will explain the approach that it is used to eliminates this redundant.

MRSimJoin algorithm [4], [5], [6] is an algorithm that is designing and implementing techniques of cloud-based Similarity Joins. This algorithm will be explained in section 2.4.

Set-Similarity Join [7] identify all pairs of high similar sets in a parallel approaches. These approaches will be explained in details in section 2.5.

Trie-join Algorithm [8] is an algorithm that is finding the similarity between two strings. This algorithm will be explained in section 2.6

The multi-sig-er method [9] is a multi-signature method based parallel entity resolution. This method will be explained in section 2.7.

In section 3, we will make a comparison between the previous algorithms according to some criteria and then analyses the results.

2. Related Work

In this section we will explain the similarity join algorithms that are mentioned in previous section.

2.1 SSJ-2R Algorithm

SSJ-2R algorithm based on MapReduce framework with speeding up the running time by factor 4.5, it is discovering every pair of objects which their similarity greater than a threshold according to some similarity function.

To solve the similarity self-join problem we can divide it into the following stages: the first stage is the signature scheme that find a compact representation of each document. The second stage is the candidate generation who's identifying potentially similar document pairs given their signatures, the third stage is the verification that is computing document similarity using measure the cosine distance, and the final stage is the indexing which is a data structure to speed up candidate generation and verification.

There are two techniques to find the signature for the document: full filtering and prefix technique. In full filtering technique the signature of the document is determined by the terms that are appearing in the document and sorted in an inverted index, it is usually used to find pairs of document which are sharing at least one term. In prefix technique the signature (S) identify which terms appearing in the document are greater than or equal to the largest integer (b(d)) of the artificial document, so if two signatures of two documents have empty intersection then they have similarity below the threshold so it is not a candidate pair. The following explained algorithms are based on prefix filtering technique.

2.1.1 Double Pass Map Reduce Prefix Filtering (SSJ2)

This algorithm sort the terms frequency in decreasing order and then the most frequent term will be discarded, this manner reduce the length of the inverted list. During the index phase inverted lists are hashed randomly to several files, this will spread the longest list uniformly over the file. The reducer receives only the contribution from terms $t>b_j$ that is mean there is no enough information to compute the similarity, so it will need two extra I/O operation to receive the two documents from distributed file system.

2.1.2 Double Pass MapReduce Prefix Filtering with Reminder File (SSJ-2R)

To avoid the remote random access in SSJ-2, *SSJ-2R* save the pruned portion for every document in a reminder file during indexing phase, this will be better regained by each node from the Distributed File System (DFS). The reminder file doesn't contain all the information needed for similarity, so it submits the document with some missing information to the reducer and define the keys produced by the map. A new map function is defined; for every couples of document in a specific inverted list, the map function generate as keys a pair of document IDs, the first one is d*i* (Least Pruned Document)LDP, and as a values the MPD (Most Pruned Document) $d_j$and the used contribution w: <<LDP*ij*, MPD*ij*>, w=di.dj>. The second Map function id defined, it takes as input the collection and the output is the documents <*i*, *di*> → (<*i*,*>, *di*), then sort the keys that are produced by the two mentioned Map functions increasingly to obtain the pair <*i*,*>

with the document d*i* followed by every document pair in which (i) is refer to the least pruned document, finally they override the group operator to determine whether the two keys are equivalent or not. based on the least pruned document, two keys <i, j> and <a, b> are equivalent if and only if i = a, so the input of the reducer is a key <*i*,*> followed by values like weights, followed by a bar of contributions that are related to the same document j being MPD, then they collect these contributions and added them to the dot product between d*i* and d_j.

2.1.3 Partitioning

The size of the reminder file will grow when collection size is increased, this will reduce the scalability of the algorithm. We can solve this problem by partitioning the reminder file, and splitting it into K (a user defined parameter) nearly equalized parts(chunks) in which the non-indexed portion of document falls into (1/ K) using Hadoop's partition function Map is done on MPD, all (i, j) pairs are mapped to a reducer in the (j/K) group, the key (i,*) and it is related value d*i* are repeated K times, so the content of LPD can be submitted to each reducer, so the reducer used to load and store chunk of reminder file.

2.2 PreJoin Algorithm

The problem of Trie-based algorithm is the large size of the active nodes, and the large edit distance threshold (τ), so when the string set is large, then the trie will be larger, to solve this problem they proposed new algorithm called PreJoin, using preorder traverse and a new generation active node method.

PreJoin algorithm: it is generating the active nodes for the next child and it is siblings, so when reach any sibling the active nodes will be available, PreJoin algorithm reorder the sibling nodes to decide the next subtrie to be visited, instead of using preorder traverse like in Trie-PathStack algorithm. Actually each trie leaf represent a string, so many nodes may represent strings that are contained by another strings, these nodes are called End of Strings (EOS) or logical variables.

PreJoin algorithm has the same steps of Trie-Path Stack algorithm except that Prejoin eliminating unnecessary active nodes during generation active nodes phase, but

Trie-Path Stack algorithm generates the active nodes then eliminate the unnecessary active nodes in pruning phase, so PreJoin algorithm proposed Novel Active Nodes Generation Method that contains three rules.

The first rule of Novel Active Nodes Generation Method is the symmetry property, this property avoids adding all the ancestors and descendants active nodes of node m into active nodes of node *n*, as follows suppose we have a current processed node *n* at depth *i*, then the number of deletion operations that is needed to transform the string n to string m, so the m nodes that lies at depth less than i- τ are the active nodes of *n*, which is the ancestors of n, the same thing for insertion operation, the summation of insertion operations that is needed to transform the string n to string m, so the m nodes that lies at depth larger than i+ τ are the active nodes of node *n*, which is the descendants of n, in addition to the node *n* that is of type EOS, the similarity algorithm search for the descendant nodes m that are of Type EOS and with edit distance τ from node n. the second rule nodes that are already traversed will not include in active nodes. The third rule when generate the active nodes of the current node from it is parent's active nodes the other siblings are not included, also theses siblings are active nodes for each other since they have an edit distance less than or equal the threshold. When a sibling is the current processing node, the generation methods take the unprocessed siblings.

Using this generation method the PreJoin becomes efficient to find all similar strings, either if they are long or short.

2.3 Pair-wise similarity

Pair-wise similarity computation is to assign objects to multiple clusters, this may produce many redundant pair's computations because more than one object may share many clusters.

Pair-wise similarity usually works as follows: for each object, regardless of the object type, at least one signature is generated, then every signature and objects that are identified the cluster are assigned to all clusters with their signatures. The map phase emits key and value for every signature (key=signature, value=object).In reduce phase the objects which are belong to the same clusters are compared with each other.

The creation of any signature is hard because you need to balance between the efficiency and the quality of data. To reduce the number of pairs the cluster size must be small as possible to increase efficiency. Small cluster size leads to lose some identical object pairs, especially in the dirty data.

The proposed algorithms (Pair-wise) reduces the pairs using MapReduce frame work as following: the signature function starts to find all signatures for all objects. In map function every object (o) is assigned to its signatures. Reduce function sort and checks the key and any pair [o1, σk(o1)], [o2, σk (o2)] even if they are disjoint. If σk(o1) ⌒ σk(o2) ≠ Φ this mean it is disjoin then we can conclude as a result that the key (k) is the least common signature value and the two objects can be compared, but if σk(o1) ⌒ σk (o2)=Φ it is joint so there is a small common signature value ќ<k and the object pair (o1, o2) is not considered for k. try to find a single signature by finding the least common signature min[σk(o1) ⌒ σk(o2)] so this responsible for the pair computation and the other pair larger than miss is discarded.

The advantages of sorting the list are: the set with small signature will simplify the prefix list and makes the comparison that determine if the two sets overlap more efficient.

Sorting the signatures in reduce phase require computing the signature of the objects again from their attributes value, this will increase the output from the map phase. The signature function will be called for every (key, value) pair, this may make the number of the objects is less than the number of calls.

Least common signature produce skew for the reduce phase because any pair shares several signature will be processed only for one signature (the smallest one), so other signatures will be considered as redundant signatures.

2.4 MRSimJoin Algorithm

MRSimJoin is a multi-round MapReduce based algorithm which is interested in the Similarity Joins for cloud systems, it is partitioning and distributing the data until the all the subsets are become small enough and then can be processed in a single node, it have implemented in Hadoop. We can use MRSimJoin algorithm in many applications such as detecting the similar images which are represented as feature vectors, and similar publications in a bibliographic database.

The input data can be given in one or more distributed files, each file may be contain records from *R* and *S* sets. Each record consist of the identification of the dataset of the record and the identification of the record in the dataset. As we mentioned before, MRSimJoin algorithm divides the data into smaller sets until each of these sets is small enough to be processed in a single-node, this called Self-Join (SJ) routine.

There are two type of partitions, Base and Window-pair partitions. Base partition include all records that are closer to a given pivot than to any other pivot. Window pair partition includes the records which is found between any two base partitions.

MRSimJoin and Quick Join algorithms are similar in the way of data partitioning, but in MRSimJoin algorithm the partitioning of the data, generation of the result links, and the storage of intermediate results are performed in a fully distributed and parallel way. There are two types of the rounds, base rounds and window-pair rounds. Base rounds are the rounds that it is used to determine the similarity links in the input data. Window-pair rounds are the Rounds that determine only the links between records which are belong to different partitions. In MRSimJoin Routine there is an intermediate directory which is used to store the partitions that will be repartitioned many times until the each partition become small and can be processed in one node. The executing of the two type of the partition (base and window-pair) MapReduce job depend on the type of the input directory and each MapReduce executed in four stages. The first stage is the Map stage, this stage responsible to divide the input data into multiple parts (chunks) and to create map tasks in multiple nodes to process them, the identical map function is called once for each input record and create single intermediate record for each base or window-pair partition.

The second stage is the partition stage, this stage is responsible to partition the intermediate data which was produced from the map tasks by calling the partition function. The third stage is the compare stage, in this stage the intermediate records which are refer to the same partition are grouped.

The last stage is the reduce stage, this stage receives the list of all records of the group and then check size of the list , if it can be processed in a single node, then calls the single-node SJ routine is called, Otherwise, all the group records are saved in the intermediate directory to be partitioning again.

2.5 Set- Similarity-Join Algorithm

Set similarity join identify all pairs of high similar sets. The large amount of pairs and sequential computations cause large execution time and more space, so in order to solve these problems, three approaches are proposed for set similarity join and perform them in parallel using MapReduce framework.

Parallel set similarity join has three stages: the first stage is to compute data statistics for good signature using Token Ordering, the second stage is to group the candidate pairs based on signature and to compute the self-similarity join using record identifications (RID)-pair generation, and the third stage is to generate the actual pairs of joined records called Record Join.

The first stage creates global ordering of the token pairs in the join column depending on their frequency, it uses one MapReduce function. In the Map function it computes the join value of each record and tokenize it, then emits each token key with the number of occurrences of it. In reduce function itcomputes the frequency for each token key and calling the tear-down function to order the tokens in memory instead of using a reducer to reduce the I/O access operations.

After scanning the input data (records), the second stage outputs RIDs pairs that are representing the records in which their join-similarity is larger than the threshold, it uses one MapReduce cycle. The map function is done with several steps: scan all input records, project them on RID and on a join attribute, tokenize them, extract the prefix depending on to the global ordering of tokens which are obtained from the previous stage, finally route the tokens to a specific reducer.

The routing strategy is done by Grouped Tokens, that means many tokens can be mapped to one key(same key),then each record generate a (key, value) pair to every group of the prefix tokens, so the group of tokens are formed by assigning tokens to group using Round Robin method. The group will be balanced because one specific group contains the sum of frequencies of token, so the replication of data is small. In the reducer function every reducer processes one or more groups, in each group the reducer find pairs which their join attribute values satisfy the similarity condition (sim (pair)>=

threshold). Finding the similarity of the candidates in group is done using Indexed Kernel strategy which use PPJoin+ index that match the probe PPJoin index with join attribute value of current candidate with a list RIDs which satisfy the similarity condition, then adding the current candidate to the PP joined index, this strategy more efficient than nested loops.

In the third stage we note that we have only pairs of RIDs, but we need actual record (reset of each record) uses two MapReduce cycles, in the first cycle each half of each pair fill the record information, and in the second cycle merge the previously filled in records.

2.6 Trie-join Algorithm

String similarity join finds similar pairs between two collections of strings. Trie-join algorithm used edit distance (ED) which is the minimum number of single character edit operations (insertion, deletion, substitution) needed to transform string “r” to the corresponding string “s” ,such that edit distance between the strings is less than a given edit distance threshold τ.

There is many algorithm studies string similarity join with edit constrains such as Part-Enum, All Pairs-ED and ED-Join, that use a filter and refine framework. in the filter stage, they generate signatures for all strings, in refine stage they find the candidate pairs and give the final results of high similar pairs, unfortunately these algorithms have disadvantages, Firstly, they are inefficient for data sets with small strings (less than 30), secondly they cannot support dynamic update of data sets, thirdly they generates large index sizes due to the large signature numbers.

Trie-Join algorithm is efficient for small strings, and developed three pruning technique to reduce number of candidate pairs, and they change this algorithm to support dynamic update and eliminate computational overhead to get high performance, they use a trie structure to index strings to reduce the storage space, since trie share the same prefixes for many strings. Trie is a tree where every path form the root to a leaf represent a string, and each node produce a character in the string, thus any node in the trie has an edit distance constrain less than the threshold is called active node.

Some traditional solutions to find similarity is to generate all string pairs and compute their edit-distance, but this method is too expensive, in order to solve this problem Trie-Join algorithm propose a new technique to reduce the computational overhead.

Subtrie pruning: based on the trie structure many strings with the same prefixes share the same ancestor nodes. Thus if we have a string in a trie, and a node is not active node for every prefix of the string, then any string under this node cannot be similar this string, so we can prune this subtrie, based on this technique they developed Trie-Search approach.

Dual Subtrie pruning: subtrie pruning is just applicable for one set of strings C, but actually the second set of strings D, is also share some prefixes, so we can make dual pruning for the two sets C, D. if we have a node x is not an active node for every ancestor in another node y, and node y is not an active node for every ancestor in node x, the strings under node x cannot be similar to any string under node y, so we can prune both of subtries.

Trie-traverse algorithm: since Trie-Search algorithm is applicable for subtrie pruning, we need a new algorithm applicable for Dual subtrie pruning, which is called Trie-Traverse algorithm, it is generate a trie index for all strings in the trie then it traverse the trie in preorder, such that it guarantees computing the parent's active nodes for every node before the node itself. Trie-Traverse computes the active nodes for each node in the trie, then when reaching a leaf node n, such that the string of the node m is belong to the active node of the leaf node, and if the node of the string is a leaf node then the algorithm get a similar string pair (n, m).

Trie-Dynamic algorithm: Trie-Travers algorithm should compute all the active nodes in the trie, but actually we don't need to compute all the active nodes of them, because of the similarity property, that says if s is an active node of t then t is an active node of s, so we can reduce the redundant computational overhead using a new algorithm, they called it Trie-Dynamic algorithm. Firstly it inserts an empty trie with only a node root, then inserts the strings in the trie. If the prefix of the string is exist it adds a leaf node an update the active nodes, otherwise if the prefix doesn't exist it inserts a new node and compute the active nodes for the new one. If s is a new inserted node, and for each node r belong to the active nodes of s, then based on symmetry property r is an active nod of r, so Trie-Dynamic algorithm update the active nodes of r by inserting node s in the active nodes of r.

TriePathstack algorithm: firstly it construct a trie for all strings, then it traverse the trie in preorder, so when TriePathStack visit a new node it generate the active nodes for the current node using virtual partial method, and it is parent's active nodes, then it pushes the node in the stack, after update the active nodes of the current nod's ancestor using the nodes away from the current node in the stack by τ (threshold value), TriePathStack check if the current node is a leaf node, then it give the similar string pair and pop the node from the stack. TriePathStack continues to repeat the previous steps until the stack is empty, thus reduce the storage space resulting from generating the active nodes using Trie-Dynamic algorithm.

Incremental similarity joins: suppose we have got the self-join similarity results of a string set R, and a new collection of string is added ΔR, it is difficult to update the self-join similarity results, so they proposed a new algorithm called Incremental similarity joins, according to this algorithm the update of the string pairs is ($s \epsilon \Delta R$, $r \epsilon R \upsilon \Delta R$), such that the edit distance is less than the threshold.

2.7 The multi-sig-er Method

Entity resolution is the main operation that it is used in data quality management, and it is used to find the value of data, in other words it is the process that find non identical duplicates and merge the identical duplicates. The problem of entity resolution is to eliminate the redundant pairs and minimize the similarity computation but at the same time preserving the accuracy of resolution. Because single signature produce a large number of objects that sharing the same key, the join computation will have a huge workload. To solve the previous mentioned problem, a multi-signature method is proposed which is based on signature blocking and parallel resolution method that supports both structured and unstructured data.

Multi-sig-er algorithm is working as following: each object is tagged with multi-signature, the same object which has many signatures can be found in different subsets, then using a blocking technique which puts the objects with the same signature in one block, this mean that one object which has many signatures may be found in different blocks, finally eliminate redundant objects by using transitive property; if we have the objects A, B, and C, object A implies object B, and object B implies object C , we can conclude by using transitive property that object A implies object C, and we don't need to match B and C since the similarity has the transitive property. By using the transitive property we can omit many redundant pairs and as a result the candidate pairs will be fewer.

3. Algorithms Comparison and Analyses

In this section we will analysis the previous similarity join algorithms in MapReduce based on some important criteria. Table 3.1 show the result of the algorithms comparison.

3.1 Preprocessing

It is a technique to make the input data easy to distribute and give an overview of the data. It needs a new stage of MapReduce so it will add new computation to the algorithm.

Set-similarity join algorithm preprocess the data using global token ordering, it scans the data and extract tokens, then compute the frequency of each token, then sort them based on frequencies, this will increase the performance.

Trie-based and pre-join algorithms are using tree data structure to organize the data, each string starting from the root to a leaf, and the intermediate nodes represent characters that are shared between more than one string, this structure reduce the space and make it easier to process the data.

SSJ_R and SSJ-R2 extract the terms of each object, and compute the frequency of each term, then sort the frequencies, the most frequent terms are discarded, this reduce the length of the inverted terms, for sure this reduce the number of candidate pairs, that will be generated from these terms.

3.2 Pre-filtering

It is a technique to eliminate some data that is not important to the join result, since the join an expensive operation to the space and the time, they try to reduce the input data to this operation.

3.3 Partitioning

It is a technique that divide the input data to multiple partitions to reduce the tasks, but this technique may generate redundant data.

Set-similarity join partition the data based on hash technique. For every record it extracts record ID (RID) and the join attribute value, then hash records to the corresponding candidate bucket, and every bucket goes to a reducer.

SSJ-R does not make partitioning, SSJ-2R partition the reminder file that contains the pruned part of the object to K (user defined variable) parts, since increasing the collection size will increase the reminder file, and since the reminder file is loaded in every node; we need to decrease its size by partition the reminder file for (K) chunks, so every reducer just need to load one chunk in the memory.

3.4 Replication

This stage is used to reduce the amount of data replication to reduce tasks that may result from partition stage.

Set-similarity join algorithm uses grouping technique that distribute all pairs which share at least one prefix, so all pairs with one prefix goes to one reducer, this guarantees pairs of record that has no chance of being similar will never go to the same reducer, so it will achieve high similarity of candidates. But grouping technique may cause multiple checks for the same record in more than one reducer, thus increase the redundant work.SSJ-2 does not has replication pairs, because of bucketing technique.

SSJ-2R has a replication, each document is replicated K times, because each part goes to one reducer, in order to retrieve the full content of the pruned document to compute the similarity.

Trie-based algorithm generates the active nodes then applying some Pruning techniques (Length Pruning, Single-branch Pruning, Count Pruning) to eliminate some the redundant pairs, but PreJoin algorithm applies Pruning during generating the active nodes to eliminate the redundant active node, so PreJoin algorithm is more efficient than Trie-based algorithm.

3.5 Load Balancing

Check if the tasks are distributing among the reducers with fairness way. Set-similarity join algorithm distribute candidates to the right reducers to minimize reducers' workload, using grouping technique. SSJ-2 and SSJ-2R uses bucketing technique, during the indexing phase it randomly hash the lists to different buckets, and every bucket goes to one mapper.

3.6 Cycles of MapReduce

How many MapReduce stages are needed using an algorithm, of course if it is one stage it is ideal, but almost all algorithms cannot make one cycle.

Set similarity join uses four MapReduce cycles, the first cycle computes the global tokens, the second cycle groups the prefixes, and the third cycle retrieves the remaining information of each record, since the last phase receive part (RID, attribute join value) of the record.

3.7 Sorting the terms

When extracting the terms of all objects, we will have signatures for them that will be used by the next tasks, if the terms are sorted it will facilitate the next tasks, so sorting How many MapReduce stages are needed using an algorithm, of course if it is one stage it is ideal, but almost all algorithms cannot make one cycle. guarantees the search time will be smaller.

SSJ-2R sorted the indexed list that makes the comparison between two overlap sets more efficient. And the same thing for SSJ-2. Set-join algorithm sorts the global frequencies.

3.8 Additional Remote I/O operation

After creating the signatures some algorithm just take these signatures to the next

MapReduce cycle to determine the keys of the objects. At the similarity calculation between two objects we need all the information of the object, so the map function must go to the distributed file system (DFS) to get the full object, thus cause an additional I/O.

Since the reducer in SSJ-2 does not receive enough information to compute the similarity, so it must go to the distributed file system to get the two documents, thus generate two remote I/O operation.

Set-similarity join algorithm bring the rest of each record using the RIDs that are generated in the previous stage, firstly it fills in the record information each half of each pair that contain a part of the record, then it brings together the previously filled in records. Thus avoid going to DFS.

SSJ-2R bring the rest of each document using the reminder file, and through MapReduce framework, so it does not go to DFS.

3.9 Computational Overhead

Redundant pairs cause an overhead since these computation are repeated many times, so it is better to eliminate this redundancy.

Set-similarity join algorithm has redundant computations, because after grouping stage some pairs goes to more than one reducer, so the similarity computation redundant in many reducer .SSJ-2 computes the similarity for every pair just once, because of applying bucketing technique, so no computational overhead. The same thing for SSJ-2R.Trie-based algorithm has a computational overhead since it have redundant active nodes. Pre-join algorithm has no computational overhead, because it generates the active node and its siblings, so there is no need to compute the active nodes for the siblings again. Pre-join algorithm use order traverse which guarantees the visited siblings will not be visited again, so if the sibling node is the current processed node its siblings that are already visited will not be visited again, by this way the computational overhead is reduced.

Table 3.1

Approach	**Pre-processing**	**Pre-filtering**	**Partitiing**	**Replication**	**Load Balancing**	**Number of Cycle of MapReduce**	**Sorting the Terms**	**Additional Remote I/O Operation**	**Computational Overhead**
Set Similarity Join	Global Token Ordering	No	Hash-Based	Grouping	Yes	4	Yes	No	Yes
SSJ-2	Global Frequency	No	Hash-Based	No	Yes	3	Yes	2	No
SSJ-2R	Global Frequency	No	Reminder file to chunks	Yes	Yes	4	Yes	No	Yes
Trie-Based	Trie Structure	No	No	No	No	N/A	No	No	Yes
Pre-Join	Trie Structure	No	No	Novel Active Node Generation Method	No	N/A	No	No	No
MRSim Join	No	No	Intermediate files	No	Yes	N/A	Yes	No	No

5. Conclusion and Future Work

The similarity join algorithms are the algorithms which are used to find the similarity between two objects who's their distance is smaller than a predefined threshold. In this survey we discussed some similarity join algorithm based on MapReduce, then we compared these algorithms (Set-Similarity join, SSJ-2R, MRSimJoin, Trie-based, PreJoin) according to some criteria that investigate the enhancement for each algorithm. We can infer from our compression that SSJ-2R is better than SSJ-2 because it has not additional I/O, in the other hand SSJ-2R has a computational overhead because it partition the reminder file. PreJoin algorithm is better than Trie-based algorithm because it eliminates the computational overhead by reducing the generated active node. Set- similarity join algorithm still has computational overhead after grouping technique, because same records might be checked for similarity in multiple reducers, but the quality for measuring the similarity is high, because the grouping technique guarantees that pairs of records which have no chance of being similar are never go to the same reducer. MRSimJoin algorithm partition the data until it can be processed on a single node, it maps the data into mappers with load balancing property which will improve the performance.

In future we will explain other similarity join algorithm, such as Fuzzy join, V-SMART-Join, Silva et al, and Top-k similarity join.

References

Baraglia, R., Morales, G. D. F., & Lucchese, C. (2010, December). Document similarity self-join with MapReduce. In *2010 IEEE International Conference on Data Mining* (pp. 731-736). IEEE.

Gouda, K, & Rashad M (2012, May). Prejoin: An efficient trie-based string similarity join algorithm. In Informatics and Systems (INFOS), 2012 8th International Conference on (pp. DE-37). IEEE.

Kolb, L., Thor, A., & Rahm, E. (2013, June). Don't match twice: redundancy-free similarity computation with MapReduce. In *Proceedings of the Second Workshop on Data Analytics in the Cloud* (pp. 1-5). ACM.

Pang, J., Gu, Y., Xu, J., Bao, Y., & Yu, G. (2014, June). Efficient Graph Similarity Join with Scalable Prefix-Filtering Using MapReduce. In *International Conference on Web- Age Information Management* (pp. 415-418). Springer International Publishing.

Silva, Y. N., & Reed, J. M. (2012, May). Exploiting MapReduce-based similarity joins. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data* (pp. 693-696). ACM.

Silva, Y. N., Reed, J. M., & Tsosie, L. M. (2012, August). MapReduce-based similarity join for metric spaces. In *Proceedings of the 1st International Workshop on Cloud Intelligence* (p. 3). ACM.

Vernica, R., Carey, M. J., & Li, C. (2010, June). Efficient parallel set-similarity joins using MapReduce. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data* (pp. 495-506). ACM.

Wang, J., Feng, J., & Li, G. (2010). Trie-join: Efficient trie-based string similarity joins with edit-distance constraints. *Proceedings of the VLDB Endowment*, *3*(1-2), 1219-

1230.

Yan, C., Song, Y., Wang, J., & Guo, W. (2015, May). Eliminating the Redundancy in MapReduce-based Entity Resolution. In *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on* (pp. 1233-1236). IEEE.