IJCTA, 9(26) 2016, pp. 161-170

An Improved Similarity Joins Using SSPS and MapReduce Technique on Large-Scale Data

P. Selvaramalakhsmi[1], S. Hari Ganesh[2] and Florence Tushabe[3]

ABSTRACT

In a day-to-day life, the capacity of data increased extremely. The growth of data will be uncontrollable in social networking sites like Facebook, Twitter, etc. In the past two years the data flow can increase in zettabyte. There are number of approaches has been developed to handle big data. However, analyzing the big data is a very challenging task now days. The current infrastructure to handle the big data is not efficient, because of large volume of data. The problems raised during the processing of big data can be resolved by using MapReduce technique. This paper proposed an efficient MapReduce technique for performing Similarity Joins between multisets using SSPS algorithm. Filtering techniques for similarity joins is used to minimize the number of pairs of entities joined, which improves the efficiency of the algorithm. This work performs the filtering techniques such as the prefix, suffix, size and positional to multisets. Algorithms are developed using Hadoop and tested using real-world Twitter data.

Keywords: Similarity Join; MapReduce; Big Data; Hadoop; 4S; Stemming.

1. INTRODUCTION

Internet Services like e-commerce websites and social networks handle huge volumes of data [1]. These services produce large volume of data from millions of users every few seconds. In the current decade data and the internet users have been increased day-by-day. Enormous growth of online applications and their users have resulted in enormous increase in the volume of data [18]. These large volumes of data are needed to be processed. It is a vital requirement to handle this massive enlargement of data over the internet. Similarity Joins are important procedure for a diverse range of applications. Some attractive applications of similarity joins such as Duplicate detection [2], [3], [4], [5] Data Cleaning [6], [7], Plagiarism Detection [8], Record Linkage [9], String Searching [10], [11], etc. The distributed processing is necessary to handle very large size data. The MapReduce framework [12] and Hadoop [13] are very popular tools which are used in this study for accomplishing these purposes.

Therefore the proposed algorithm focused on multisets corresponding to the pair must be grouped together in the same reduce instance to perform the similarity join. To create an efficient and scalable MapReduce style work flow is tricky and challenging task.

Some of the available MapReduce similarity join algorithms [15], [16], have included no filtering techniques. However, some [6], [17] have tried to incorporate filtering techniques. In the second cases, despite the attempt to incorporate filtering techniques, large quantity of data is produced causing I/O and network bottlenecks which has resulted in inefficiency and poor scalability.

[1] Assistant Professor, International University of East Africa, Kampala, Uganda, *E-mail: lakshmibhabuu@gmail.com*

[2] Assistant Professor, Department of Computer Science, H. H. The Rajah's College, Madurai Road, TamilNadu, India, *E-mail: hariganesh17@gmail.com*

[3] Associate Professor, Directorate of Engagement Research and Innovation, UTAMU, Kampala, Uganda, *E-mail:floratush@gmail.com*

The main contributions of this research work are listed below:

1. An efficient MapReduce Algorithm for executing Similarity Joins between multisets is presented which is also appropriate to sets nand vectors.
2. Filtering techniques developed for Similarity Join in sets are extended for application to multisets, including prefix and suffix truncation using stemming [27], [28], size [20] and positional [19] filtering.
3. All filtering techniques are integrated into the MapReduce framework to minimize the I/O, network bottlenecks and the computation.
4. A new method for enhancing the scalability of the proposed algorithm is created.

The remainder of this paper is structured as follows: in Sect. 2, the Background information necessary to understand this research work is discussed. In Sect. 3, the proposed algorithm is illustrated. In Sect. 4, results and Comparison of the proposed algorithm are presented. In Sect. 5, the conclusion of this research paper and future work is presented.

2. BACKGROUND

This section initiates the preliminary background essential for creating the proposed algorithm including the fundamental concepts of MapReduce model, Hadoop features, Multisets, Measures of Similarity Join and Programming Model of MapReduce in Parallel Computing algorithms.

2.1. Fundamental concepts of MapReduce model

MapReduce framework has the syntax of map function and reduces functions. MapReduce technique permits distributed procedure for Map/Reduction functions. MapReduce [13] is an easy programming model for dispensation large volume of data sets in parallel. The MapReduce Framework [12] was developed for large-scale parallel and distributed processing. The framework handles the parallelization, fault tolerance, load balancing over a cluster of machines, data transfers, etc.

The data are written to and read from a Distributed File System (DFS). A programmer-defined Map Function processes a key/value pair input record and sends out a list of intermediate key/value pairs. A programmer-defined Reduce Function accepts a list of values equivalent to an identical intermediate key, and processes it to throw out a list of key or value pairs. Next, the intermediate keys are partitioned to the right reducer by the partitioning function. Additionally, it is ensured that the partitioned records are sorted and forms the Shuffle Phase.

The default partitioning function is hash (key) mod R, where R is the number of reducers or partitions. The programmer has the elasticity of choosing his own routine partitioning function as well. The programmers do not need to be worried about the execution particulars of parallel processing in MapReduce framework because its programming model is automatically parallelized, so that the programmers can write only two functions: map and reduce [21]. The MapReduce Model is represented in Figure 1.

The map function reads the data input in parallel and distributes the output data to the reducers. The reduce function receives output from the map function and then produce a list containing all the values of output with that key [14].

2.2. Hadoop features

Hadoop [13] is a commonly used open source implementation of MapReduce algorithm. Hadoop offers a feature called Distributed Cache. Using this feature, the framework copies the desired files, archives, jars, etc to the local hard disk of the slave node before any task is executed on that node. Secondary sorting can

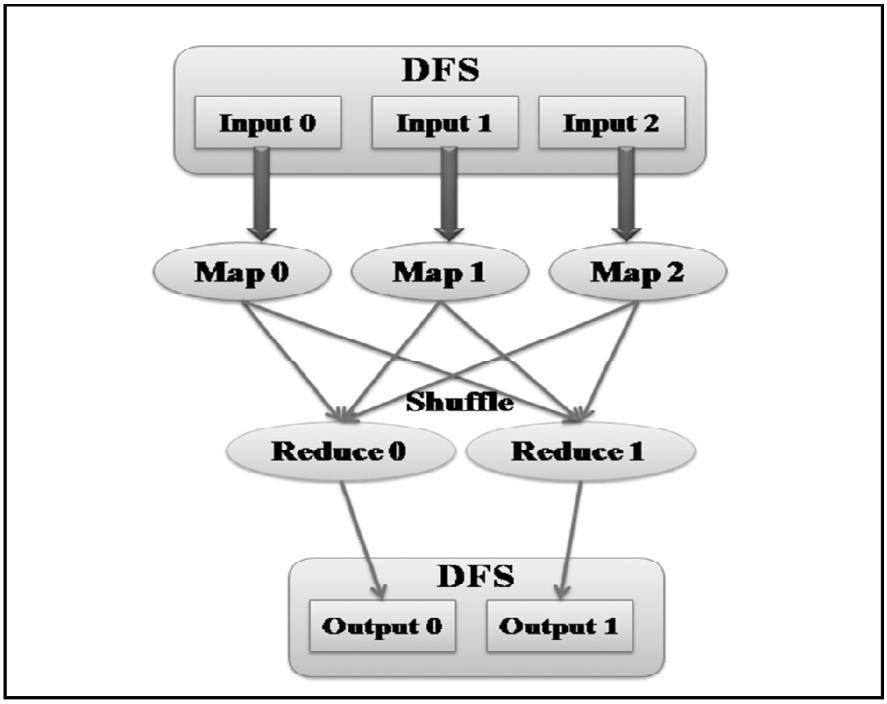

Figure 1: MapReduce Model

be achieved in Hadoop. Keys of the records can be composite into primary key and secondary key using Hadoop. Records are sorted based on the primary key during the Shuffle Phase. Secondary sorting means that records with the same primary key are arranged based on the secondary key. It is possible with the help of a custom partitioning function, sorting comparator and grouping comparator. Next, the custom practitioner makes sure that all the records with the same primary key achieve the same reducer. The sort comparator makes sure that all the records with the identical primary key are sorted based on the secondary key. The grouping comparator makes sure that all records with the similar primary key reach the same reduce function in a reducer. The records reached in a reduce instance are sorted based on the primary key. The records with the same primary key are sorted based on the secondary key.

2.3. Multisets and Similarity Measures

In this research work, we considered multiset, multiset union, multiset intersection, and multiset size.

Multiset: Multiset (Mi) is a generalization of the concept of a set which allows multiple instances of the multiset's elements. For example, {a, a, b} and {a, b} are different multisets although they are the same set. The set indicator function of a normal set is generalized to the multiplicity function for multisets. The set indicator function of a subset A of a set X is the function.

$$1_{\mathrm{A}} : \mathrm{X} \rightarrow \{0, 1\}$$

defined by

$$1_A(x) = \begin{cases} 1 \; if \;\; x \in A, \\ 0 \; if \;\; x \notin A \end{cases} \tag{2}$$

Multiset Intersection: The set indicator function of the intersection of sets is the minimum function of the indicator functions

$$1_{A \cap B}(x) = \min\{1_A(x), 1_B(x)\} \tag{3}$$

Multiset Union: The set indicator function of the union of sets is the maximum function of the indicator functions

$$1_{A\cup B}(x) = \max\{1_A(x), 1_B(x)\} \quad (4)$$

Multiset Size: Multiset Size $|M_i|$ is the sum of the frequencies of the data elements (d_k) of all the elements present in M_i.

$$|M_i| = \Sigma_{1\le k\le |D|} f_i(d_k) \quad (5)$$

Multiset Element Position: For every multiset M_i, its elements are ordered by a global ordering, that is common for all the multisets. The position of $m_{i,j}$ in M_i denoted by Pos (mi, j), is the sum of the frequencies of the data elements of all the elements in M_i.

$$Pos(i, j) = \Sigma_{1\le k\le j-1} f_i(d_k) \quad (6)$$

Similarity Measures: Similarity Measure is a real-valued function that quantifies the similarity between two objects or strings. The similarity measure can be performed based on stemming technique [27], [28].

2.4. Serial Similarity Join Algorithms

In order to progress the efficiency, the algorithms in the literature have focused on reducing the number of candidate pairs using which Similarity Joins have to be achieved. Using this technique the data elements are hashed, therefore the similar elements are hashed to the same bucket with a high probability [24]. LSH technique supported on min wise independent permutations to compute the Jaccard Similarity, for which duplicate web documents has been detected [25].

An inverted index [26] maps an element to the entities that contain it. As a substitute of comparing all the entities in a collection, it helps to compare the entities that at least contain a common element. An inverted index-based algorithm is used to index every set element to generate candidate pairs for similarity computation with optimizations based on threshold information, sorting and clustering based techniques [29].

A signature-based technique namely PartEnum, uses two sets which share a signature in common if their Hamming Distance is less than a particular value (k), was developed by [20]. Furthermore, similarity is estimated for sets sharing at least a signature. Prefix filtering technique is used for generating candidate pairs using candidates was proposed by [18]. All Pairs Algorithm to calculate the similarity of all the pairs of related candidates based on the threshold value during indexing to reduce the candidate pairs produced was proposed by [25]. A prefix filtering-based algorithm, where top 'k' similar elements are returned, if the similarity threshold is unknown, was developed by [30]. A state-of-the-art technique for detecting duplicate documents, where prefix filtering was combined through further filtering techniques like the positional and suffix filtering for sets, named as PPJoin+, was proposed by [19].

2.5. Parallel Similarity Join Algorithms

A MapReduce algorithm for finding traffic load balancing proxies and anomalies in internet with similarity joins was proposed by [31]. Initially, this algorithm calculated the size of every multiset. In the second stage, all the multiset elements have indexed in the Map Phase and the MIDs (Multiset ID) that share a common multiset element are grouped in the Reduce Phase and possible candidate pairs have generated. Since every multiset element is indexed without incorporating any filtering technique, humongous number of pairs will be generated. They must be written to the DFS by the Reduce Phase, causing I/O inefficiency. These humongous numbers of pairs are read by the Mappers and passed through the network causing congestion and inadequacy, which is not a scalable approach. However, this algorithm fails to incorporate other filtering techniques such as suffix filtration, affix filtration, etc.

3. PROPOSED ALGORITHM

The proposed algorithm has three stages, each of which consisting of a Map and a Reduce Phase. Stage I is performed the Stemming Technique [27], [28] using this technique the prefix elements and suffix elements of the multisets are indexed separately. Records sharing the same prefix data element are grouped separately as well as suffix data elements are grouped separately to generate the possible candidate pairs for which similarity joins are to be performed. Next, the Size Filtering is performed to reduce the pairs. In Stage II, Positional Filtering is applied to the pairs. In Stage III, Similarity Joins are performed using both Intersection and Union. The Proposed SSPS (Stemming, Size Filtration, Positional Filtration and Similarity Join) is show in Figure 2.

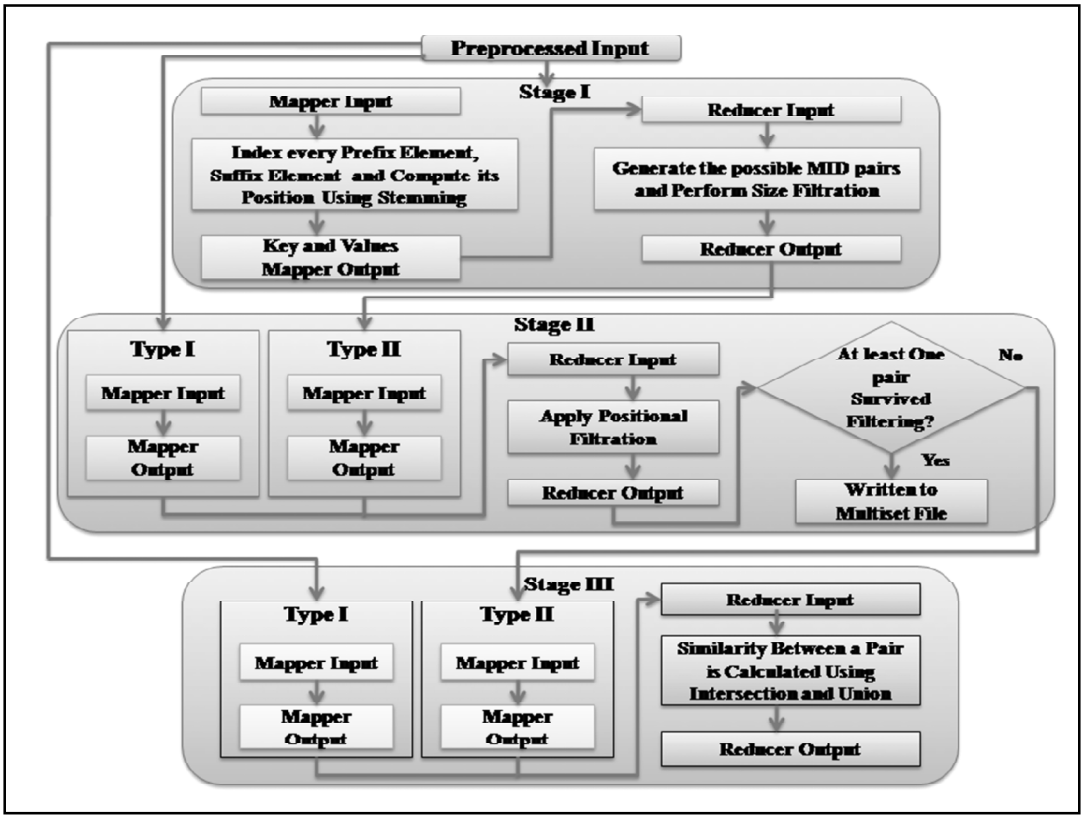

Figure 2: The Proposed SSPS Algorithm

The three stages are detailed in the following sections.

3.1. Stage I-Map Phase

The preprocessed input to the Map Phase consists of various records, each containing the Multiset ID, Multiset (M_i), followed by the elements of M_i. The elements of M_i are arranged based on the increasing order of the global frequency and the least frequent being the first. Each Mapper calculates the prefix size and suffix size of the multiset in the input record. For every element, the prefix of the multiset M_i with data element d_k, the following information such as MID, M_i, the size of Mi | Mi | and d_k's frequency in M_i , $f_i(d_k)$ and position of the element in M_i , Pos($m_{i,k}$), are sent as the Mapper value. The key being $d_k.M_i$ is calculated by summing up the frequencies of the data elements of the multiset elements present in M_i as given by the expression 5. Pos($m_{i,k}$) of a multiset element, $m_{i,k}$ is calculated by summing the frequencies of the data elements of the multiset is represented.

3.2. Stage I-Reduce Phase

At each reducer, the records which share the same prefix data element and suffix data element in d_k, as the key, are grouped separately. Using stemming technique, prefix and suffix elements in multisets grouped. If two multisets have a common prefix data element, which are potential candidates of being similar are grouped. Therefore, all the possible MID pairs that share the same d_k are generated. To reduce this number further, the suffix filtration [27] and the size filtering technique [20] are applied, which gives effective pruning results. The size filtration technique is applied using the size information sent with every record. For every MID pair, $\{M_i , M_j\}$ and threshold t, if the size filtering condition, $| M_j | \geq t^* | M_i |$, is satisfied, it passes the filter; otherwise it is pruned. For every MID pair, $\{M_i, M_j\}$ that survives size filtering, the frequency of d_k, size and position information of both M_i and M_j are appended and sent as the reducer output.

3.3. Stage II-Map Phase

Stage II-map phase consists of two types of Mappers

Type I Mappers: The preprocessed input of Stage I-Map Phase, where each records consisting of the MID, M_i and its elements, are read here and sent as output with $\{M_{i,m}\}$ as the key and the elements of M_i as the value. Here, the 'm' in the key denotes that it is a record containing the multiset elements. It is called multiset records. The proposed algorithm customized with the Partitioning, Grouping, and Sorting. So that in each reduce instance records will be grouped based on the MID, M_i, the primary key and sorted based on the secondary key.

Custom Partitioning: A multiset record has the composite key $\{M_{i,m}\}$, where M_i is the primary key and m is the secondary key. An MID pair record has the composite key $\{M_i, M_j\}$, where the primary key is M_i and M_j is the secondary key. Records are partitioned based on the primary key. Both types of records, for which the primary key and M_i is the same, are partitioned to the same reducer.

Custom Grouping: Custom Grouping ensures that records that have the same MID, M_i as the primary key reach the same instance. Each reduce instance thus pertains to a unique MID M_i.

Custom Sorting: Sorting is based on the secondary key so that the record containing the multiset elements arrives first in an instance and is followed by the MID Pair records.

Type II Mappers: The records obtained from the output of Stage I-Reduce Phase are read. These records relate to MID pairs and are denoted as MID Pair records. The output key is the MID pair $\{M_i, M_j\}$, which comprises of the frequency of d_k, size and position information of both M_i and M_j, to assist positional filtering in the Reduce Phase. In the Stage I-Reduce Phase, the records which sharing the common prefix element can be joined together and sent as output. Similarly, the records sharing the common suffix elements can also be appended together and throw as output. Candidate pairs can be generated by the Type I-Mappers of Stage II. This optimization technique is also implemented by the SSJ-2R algorithm [6].

3.4. Stage II-reduce phase

The records which have the same M_i as primary key are grouped in the same instance. These include the multiset record corresponding to M_i and the MID Pair records with the same M_i as their primary key. In every reduce instance, the multiset record with key, $\{M_{i,m}\}$, arrives first. Following that, the MID Pair records arrive. The MID Pair records arrive in a sorted order based on their secondary key M_j. The MID Pair records that pertain to the same $\{M_i , M_j\}$ pair are grouped together and positional filtering is applied. Every unique pair $\{M_i , M_j\}$, that survives positional filtering is sent as output. If there is at least one pair that survives positional filtering, MID M_i and its elements are written to a file named as the Multiset File. An important aspect of the Stage II-Reduce Phase is positional filtering. Positional filtering is the technique of filtering pairs of sets based on the positional information of the overlapping token between the sets

3.5. Stage III-map phase

Stage III-map phase consists of two types of Mappers.

Type-I Mappers: It reads the preprocessed input of Stage I-Map Phase, where every record consists of the MID, M_i and the elements of M_i. These records are sent as output with $\{M_{i,m}\}$ as the key and the elements of M_i as the value.

Type-II Mappers: It reads the outputs of Stage II-Reduce Phase which are the MID Pairs. Each record pertains to a pair, $\{M_i, M_j\}$. In the previous stage, the elements of every multiset, M_i, having at least one pair surviving positional filtering, having Mi as the first of the pair is written to the Multiset File. So, the elements of multiset M_i can be retrieved from the Multiset File in the Reduce Phase, but we cannot retrieve elements of multiset M_j from it. To solve this problem, the record is reversed and sent out from the Mapper with the $\{M_j, M_i\}$ as the key and M_j as the value.

3.6. Stage III-reduce phase

The Multiset File is loaded into the memory by every reduce node. Partitioning, Grouping, and Sorting are done in the same way as Stage II. Same as the Stage II, records that have the same M_i as the first part of the key arrives at each reduce instance. Every MID pair $\{M_i, M_j\}$ gets the elements of M_i from the multiset record that arrives to the same instance, and looks up the Multiset File for the multiset elements of M_j. Both the multisets, M_i and M_j of a pair are available and the similarity between them is computed. Similarity between a pair can be computed using stemming technique developed by [27], [28].

4. RESULTS AND COMPARISON

The experiments have been conducted on a Hadoop cluster with 51 virtual nodes and one additional node for handling the Hadoop master daemons. Each node has a memory allocation of 8 GB, a single 2.8 GHz CPU, 64bit Operating Sytems and 40 GB of disk space. The simulations were completed using 60 GB of raw twitter date in the JSON format. These data were preprocessed to remove stop words and get the desired form. Each record containing a user's ID and a multiset of the words of the tweets which were sent by the user. Similarity Joins are performed between the multisets of various twitter users to determine their similarity. Table 1 shows the running times of SSS and SSJ-2R algorithms for varying number of input records and the corresponding performance improvement (% reduction in runtime) in SSS in comparison to SSJ-2R, for thresholds 0.7.

The graph shown in Figure.3 are plotted by applying the algorithms on varying number of input records and finding the corresponding running times for similarity thresholds of 0.7. As we can see from the table and the graph, the running times increase with the increase in dataset, as well as with decrease in thresholds, as a greater number of candidate pairs will be generated and joined. A speed up of up to 40% is obtained in SSPS on comparison to SSS. As a result of incorporating the various filtering techniques, SSPS algorithm minimizes the number of pairs generated and joined and outperforms SSS and SSJ-2R.

5. CONCLUSION

An efficient MapReduce algorithm for performing Similarity Joins between multisets named SSPS is proposed. This algorithm consists of three MapReduce stages that show to extend the prefix and suffix filtration, size filtration and positional filtration technique to multisets, efficiently incorporating them in a strategic sequence in the MapReduce framework. Some of the existing technique is not scalable when the set size is large. This is because of the excessive replication performed causing network congestion.

Table 1
Running Time of SSPS, SSS and SSJ-2R

Number of Records	*Algorithm*	*Running Time (s)*
7281	SSPS	321
	SSS	558
	SSJ-2R	1763
11306	SSPS	655
	SSS	897
	SSJ-2R	2905
14336	SSPS	1211
	SSS	1620
	SSJ-2R	4504
16244	SSPS	1710
	SSS	1855
	SSJ-2R	6480

Figure 3: Running Time of SSPS, SSS, SSJ-2R

It was shown that SSPS utilized the size information of the pairs to perform size filtering to prune pairs further. Size filtering reduces the number of pairs transferred from the Mappers through the network to the Reducers in the Stage II. In the Stage II Reduce Phase, SSPS performs Positional Filtering to filter candidate pairs based on the position information of the multiset prefix element as well as suffix elements. The SSPS performs the positional filtering for aggregate the candidate pairs. It is possible for SPSS to perform all the filtering techniques and it passes the size and positional information with the indexed prefix elements and suffix elements in the first stage, thereby strategically perform various filtering in the successive stages.

Applying these filtering techniques drastically reduced the pairs to be joined, leading to increased computational efficiency. Another important aspect of SPSS algorithm is that it performs the joins in a scalable manner. This paper proposed a very rational technique to enhance the scalability of the algorithm to deal with the condition where the Multiset File used for performing the joins grows too large to be held in the memory. The algorithm splits the Multiset File which cannot be held in the memory into chunks of k

files, each of which can be held in the memory, and performs the joins in k waves. The specialty of this technique is it makes the algorithm scalable in two different types of scenarios. The first is scalability with regard to an increase in Multiset File size. The second is scalability with regards to an increase in the number of nodes in the cluster. Through experimentation and analysis, it was shown that 4S has increased I/O, network, and computational efficiencies in comparison with the other algorithms and was shown to outperform the competing SSJ-2R algorithm by over 60 % and SSS algorithm by over 40%, by testing them on Twitter Dataset and PAN Documents Collection Dataset

REFERENCES

[1] Bakshi, K., "Considerations for Big Data: Architecture and Approach", In: IEEE Aerospace Conference, pp.1-7,USA, 2012.

[2] Fetterly D, Manasse M, Najork M (2003) On the evolution of clusters of near-duplicate web pages. J Web Eng 2(4):228–246.

[3] Henzinger M (2006) Finding near-duplicate web pages: a large-scale evaluation of algorithms. In: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval. ACM, New York, pp 284–291.

[4] Sarawagi S, Bhamidipaty A (2002) Interactive deduplication using active learning. In: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, New York, pp 269–278.

[5] Xiao C,WangW, Lin X, Yu JX,Wang G (2011) Efficient similarity joins for near-duplicate detection. ACM Trans Database Syst (TODS) 36(3):15.

[6] Baraglia R, De Francisci Morales G, LuccheseC(2010) Document similarity self-joinwith mapreduce. In: 2010 IEEE 10th International Conference on Data Mining (ICDM), IEEE, pp 731–736.

[7] Elsayed T, Lin J, Oard DW (2008) Pairwise document similarity in large collections with mapreduce. In: Proceedings of the 46th annual meeting of the association for computational linguistics on human language technologies: short papers. association for, computational linguistics, pp 265–268

[8] Hoad TC, Zobel J (2003) Methods for identifying versioned and plagiarized documents. J Am Soc Inf Sci Technol 54(3):203–215.

[9] Winkler WE (1999) The state of record linkage and current research problems. In: Statistical Research Division, US Census Bureau, Citeseer.

[10] Hadjieleftheriou M, Chandel A, Koudas N, Srivastava D (2008) Fast indexes and algorithms for set similarity selection queries. In: IEEE 24th International Conference on Data Engineering, 2008. ICDE 2008. IEEE, New York pp 267–276

[11] Henzinger M (2006) Finding near-duplicate web pages: a large-scale evaluation of algorithms. In: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval. ACM, New York, pp 284–291.

[12] Dean J, Ghemawat S (2008) Mapreduce: simplified data processing on large clusters. Commun ACM 51(1):107–113

[13] The Apache Software Foundation (2014) Hadoop. URL: http://hadoop.apache.org

[14] Hadoop :open source implementation of MapReduce, <http://hadoop.apache.org/mapreduce/>.

[15] Elsayed T, Lin J, Oard DW (2008) Pairwise document similarity in large collections with mapreduce. In: Proceedings of the 46th annual meeting of the association for computational linguistics on human language technologies: short papers. association for, computational linguistics, pp 265–268.

[16] Metwally A, Faloutsos C (2012) V-smart-join: a scalable mapreduce framework for all-pair similarity joins of multisets and vectors. Proc VLDB Endow 5(8):704–715.

[17] Vernica R, Carey MJ, Li C (2010) Efficient parallel set-similarity joins using mapreduce. In: Proceedings of the 2010 ACM SIGMOD international conference on management of data. ACM, New York, pp 495–506.

[18] Chaudhuri S, Ganti V, Kaushik R (2006) A primitive operator for similarity joins in data cleaning. In: Proceedings of the 22nd international conference on data engineering, 2006. ICDE'06. IEEE, New York, pp 5–5.

[19] Xiao C,WangW, Lin X, Yu JX,Wang G (2011) Efficient similarity joins for near-duplicate detection. ACM Trans Database Syst (TODS) 36(3):15.

[20] Arasu A, Ganti V, Kaushik R (2006) Efficient exact set-similarity joins. In: Proceedings of the 32nd international conference on Very large data bases, VLDB Endowment, pp 918–929.

[21] Gu, R., Yang, X., Yan, J., Sun, Y., Wang, B., Yuan, C. & Huang, Y. (2014) SHadoop: Improving MapReduce Performance by Optimizing Job Execution Mechanism in Hadoop Clusters. Journal of

[22] Parallel and Distributed Computing, 74(3), 2166-2179.

[23] White T (2009) Hadoop: the definitive guide: the definitive guide. O'Reilly Media.

[24] Indyk P, Motwani R (1998) Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of the thirtieth annual ACM symposium on theory of computing. ACM, New York, pp 604–613.

[25] Bayardo RJ, Ma Y, Srikant R (2007) Scaling up all pairs similarity search. In: Proceedings of the 16th international conference on World Wide Web. ACM, New York, pp 131–140.

[26] Ricardo BY *et al*. (1999) Modern information retrieval. Pearson Education India, Delhi.

[27] M.Kasthuri and S.Britto Ramesh Kumar, "A Framework for Language Independent Stemmer Using Dynamic Programming", International Journal of Applied Engineering Research Volume 10, Number 18, pp 39000-39004, ISSN 0973-4562, 2015.

[28] M. Kasthuri and Dr. S. Britto Ramesh Kumar, "Multilingual Phonetic Based Stem Generation", In: Proceedings of the Second International Conference on Emerging Research in Computing, Information Communication and Applications (ERCICA-2014), ELSEVIER Science::Technology India, ISBN: 9789351072607, Volume.1, pp.437-442, 01-02 August, Bangalore, India, 2014.

[29] Sarawagi S, Kirpal A (2004) Efficient set joins on similarity predicates. In: Proceedings of the 2004 ACM SIGMOD international conference on management of data. ACM, New York, pp 743–754.

[30] Xiao C, Wang W, Lin X, Shang H (2009) Top-k set similarity joins. In: IEEE 25th international conference on data engineering, 2009. ICDE'09. IEEE, New York, pp 916–927.

[31] Metwally A, Faloutsos C (2012) V-smart-join: a scalable mapreduce framework for all-pair similarity joins of multisets and vectors. Proc VLDB Endow 5(8): 704–715.