Load-Balancing the Distance Computations in Record Linkage

Dimitrios Karapiperis Vassilios S. Verykios

Hellenic Open University
School of Science and Technology
Patras, Greece
{dkarapiperis, verykios}@eap.gr

ABSTRACT

In this paper, we propose a novel method for distributing the distance computations of record pairs generated by a blocking mechanism to the reduce tasks of a Map/Reduce system. The proposed solutions in the literature analyze the blocks and then construct a profile, which contains the number of record pairs in each block. However, this deterministic process, including all its variants, might incur considerable overhead given massive data sets. In contrast, our method utilizes two Map/Reduce jobs where the first job formulates the record pairs while the second job distributes these pairs to the reduce tasks, which perform the distance computations, using repetitive allocation rounds. In each such round, we utilize all the available reduce tasks on a random basis by generating permutations of their indexes. A series of experiments demonstrate an almost-equal distribution of the record pairs, or equivalently of the distance computations, to the reduce tasks, which makes our method a simple, yet efficient, solution for applying a blocking mechanism given massive data sets.

1. INTRODUCTION

The process of merging disparate data sets in order to identify records that refer to the same real world entity is known as the record linkage, or the entity resolution, or the data matching problem [2]. Basically, record linkage consists of two steps, namely the searching/blocking step and the matching step. In the first step, potentially matched pairs are formulated while in the second step these pairs are compared. For example, the standard blocking mechanism [3], which has been used extensively in the last decades, inserts all records that exhibit the same value in a chosen field(s) (attribute) into the same block. During the second step, the record pairs formulated are compared by using a metric so that their distance is determined. For example, we can use edit distance [10] for comparing string values and determining their proximity. The distances of the corresponding fields are used as input to a rule which classifies a record pair as a matched pair or as a non-matched pair.

Due to the fact that distance computations are expensive, the solutions proposed for the searching step focus on minimizing the number of record pairs that are brought together for comparison. By doing so, the time required for the two steps, described above, is reduced significantly since a large amount of non-matched records pairs, which entail useless distance computations, are not considered for comparison. These solutions (see a detailed survey in [3]), though effective for data sets of small or moderate size, exhibit a considerable overhead in terms of performance when applied to massive data sets. Exactly for this reason, one can enable a scalable solution for the blocking mechanism applied by utilizing a Map/Reduce system.

Generally speaking, Map/Reduce is a framework for performing parallel computations on a massive data set by using a large number of compute nodes. A Map/Reduce system consists of two main phases. The first is the map phase where the corresponding tasks take as input a set of data which is broken down into key/value tuples. Next, those tuples are forwarded to the tasks of the reduce phase where the tuples are aggregated by the same key. Then, summary operations are usually performed using the aggregated values of each key. As the sequence of the name Map/Reduce implies, the reduce phase is always performed after the map phase. Adjusting this model to the blocking step of record linkage, one can say that a key/value tuple consists of the blocking key as the key and the record as the value. Thus, during the reduce phase, record pairs are formulated by the aggregated records in each block and their corresponding distances are computed. All tuples, aggregated by a blocking key, are forwarded to a certain reduce task chosen (e.g., randomly or by hashing the blocking key) by the Map/Reduce system. This actually means that although each reduce task may have received record pairs corresponding to a certain number of blocks, being equal for all the reduce tasks, the actual load processed by each task, which is the volume of the distance computations, may exhibit large variations. As a result, some reduce tasks are overloaded by a large number of tuples and at the same time some others are underutilized due to the much smaller number of tuples forwarded. By assuming the standard blocking mechanism with the *LastName* field as the blocking key, values such as "Smith", or "Johnson", or "Williams" [11] will formulate overpopulated blocks while at the same time there will be blocks corresponding to less frequent values. This imbalance problem is known as the data skew issue of the reduce phase, which has as a consequence the poor utilization of computational resources such as space and time consumption.

In this paper, we propose a solution for the fair distribution of distance computations, which can be considered as the load in the record linkage context, by using a Map/Reduce system. Our method utilizes two Map/Reduce jobs where the output of the first job is used as the input to the second job. During the first job, record pairs are formulated by applying a blocking mechanism while in the second job we distribute these pairs to the available reduce tasks for performing the distance computations. The distribution occurs by using these reduce tasks repetitively in allocation rounds. In each round, each record pair is assigned an index which represents a reduce task and is chosen randomly and uniformly. Then, each record pair is forwarded to the reduce task specified by that index so that the corresponding distance computation can be performed. This index is not used again until all indexes in the current round are exhausted. At that point, a new allocation round of record pairs to the reduce tasks begins. We provide both a theoretical analysis and a variety of experimental results that illustrate the efficiency of our scheme in minimizing the data skew in the Map/Reduce framework during the distance computations of record pairs formulated by a blocking mechanism.

The structure of the paper is organized as follows. Related work is described in Section 2. An outline of the processing tasks, which compose a Map/Reduce job, is given in Section 3 as well as we describe the data skew issue which is responsible for the imbalance problems in Map/Reduce. Our method is illustrated in Section 4 along with a theoretical analysis. Experimental results, including comparisons to other state-of-the-art methods, are presented in Section 5. Conclusions and ideas for future extensions are discussed in Section 6.

2. RELATED WORK

The Map/Reduce paradigm has been extensively used as a scalable tool for performing similarity joins in massive data sets. For example, Vernica et al. [13] present a framework of answering set similarity join queries in parallel by partitioning the data across nodes in order to balance the workload and minimize the need for replication. Blanas et al. in [1] demonstrate the performance of several join algorithms while in [9], Okcan and Riedewald implement arbitrary joins by using a single Map/Reduce job. They perform an analysis by sampling records in order to derive input statistics and minimize the job's completion time.

Especially in record linkage, the proposed solutions, which focus on the reduce phase data skew issue, employ a deterministic approach by running a separate Map/Reduce job that performs an analysis of block cardinalities on the considered data sets. This analysis is summarized in terms of a profiling data structure which could be materialized by a matrix utilized by the subsequent Map/Reduce jobs that perform the blocking and the matching step.

Authors in [7] calculate block cardinalities and store them in a matrix which is then shared by the subsequent Map/Reduce jobs. However, this method falls short during the matrix construction step given massive data sets. A scalable solution is presented in [14] where two low-memory consumption algorithms use block cardinality estimations provided by a sketch [4] in order to distribute the load accordingly. One of these algorithms is evaluated in Section 5.

SkewTune presented in [8] is an innovative solution for the reduce phase skew issue and works as follows. Whenever a node in the cluster is identified as idle, the task with the greatest expected remaining processing time is chosen and its unprocessed input data is redistributed so that all nodes are fully utilized. This redistribution though may incur considerable latency especially in cases where large chunks of data should be relocated.

In [6], authors propose a strategy for de-duplicating a redundant blocking mechanism, based on the Locality-Sensitive Hashing technique [5], by utilizing two Map/Reduce jobs. The first job formulates the record pairs while the second one eliminates the duplicate pairs.

3. PROBLEM FORMULATION

For illustration purposes and without loss of generality, we assume that two data custodians, namely Alice and Bob, are about to merge their data sets, A and B respectively, in order to find similar records which might represent common entities. We also assume a trusted third party, namely Charlie, who offers linkage services by utilizing a robust computational infrastructure which is capable of handling massive data sets. The data custodians submit to Charlie these data sets, which follow a common schema, and assign to him the task of identifying the matched record pairs.

Charlie, in order to avoid comparing each record of A with each record of B ($A \times B$), generates record pairs by applying the standard blocking mechanism as follows. A blocking key for each record is specified as the value of a chosen field, or fields, of the common schema. All records that exhibit the same blocking key value are inserted into the same block. The record pairs formulated within each block are compared in order to determine their distance. However, when we deal with massive data sets these record pairs could be in the order of millions or billions. For this reason, Charlie maintains a cluster of commodity hardware with a distributed file system that runs on top of it in order to leverage scalability of the required computations. The data sets are segmented by the distributed file system into equal subsets which are then distributed to the compute nodes of the cluster.

During processing, specialized tasks run on each node that perform computations by using only the local subset of records on each node. By assuming N compute nodes, the data sets A and B are segmented into N subsets which are distributed to the compute nodes. We have to note, that each such subset consists of records of both data sets A and B. On each node, a task is initiated, that is M_i, which uses for each record the value of the blocking key K and formulates a tuple $<K, V>$ where V could be the record itself or an Id that references this record [1]. Then, on each node a task denoted by P_i receives as input the output of the local M_i, which is a volume of $<K, V>$ tuples, and for each tuple it generates an integer value according to the key K. This integer value represents the task of the following phase to which this tuple should be forwarded. Therefore, pairs exhibiting the same key K will be forwarded to the same task. Finally, this latter family of tasks receive the forwarded pairs, aggregate those pairs which exhibit the same K, and then perform the distance computations. All these tasks run in three distinct phases and they are coordinated by the Map/Reduce system

[1]The *Id* field plays the role of an identifier for uniquely identifying each record in a data set. Therefore, records could be retrieved by a distributed database running together with a Map/Reduce system such as Apache HBase `http://hbase.apache.org/`.

as summarized below.

i *Map phase.* Each map task M_i, where $i = 0, \dots, N_M - 1$ by assuming N_M map tasks in total, builds the blocking key K of each record at hand and emits it to the partitioner task along with value V which is the *Id* of the corresponding record. This tuple is denoted by $<K, V>$.

ii *Distribution of keys.* The partitioner tasks P_is, where a P_i corresponds to an M_i, control the distribution of each $<K, V>$ tuple to the reduce tasks. Tuples with the same key will be forwarded to a single reduce task. An efficient partitioning has as a result the least possible idle time for each reduce task.

iii *Reduce phase.* Each reduce task R_j, where $j = 0, \dots, N_R - 1$ by assuming N_R reduce tasks in total, processes its load of $<K, [V]>$ tuples distributed to it by the P_is. The notation $[V]$ denotes the vector of values aggregated for this blocking key K.

This workflow constitutes a Map/Reduce *job*, denoted by J. Complex computational problems though cannot be solved by a single job instead should be broken down into simpler subproblems where each subproblem should correspond to an individual job. Thus, a pipeline of chain jobs is executed where the output of a job is used as the input to the next job in line.
Tasks in each phase run independently and by no means do they communicate with one another. Although, this independence makes the whole system quite robust, because the failure of a task does not affect the execution of the other tasks, the distribution of load by the P_is to the R_js becomes a non-trivial issue. By taking into account that (a) the number of the formulated record pairs in certain blocks may be excessively large and (b) the independence of each R_j, we will end up with some R_js, which will receive the load of the overpopulated blocks and will dominate the matching step, and a number of underutilized R_js with much fewer records forwarded. This imbalance is commonly known as the data skew issue of the reduce phase and results in severe performance degradation of the record linkage process.

4. A PERMUTATION-BASED LOAD-BALANCING METHOD

Our Permutation-Based Load-Balancing method (PBLB) utilizes two Map/Reduce jobs, namely J_1 and J_2. Job J_1 formulates the *Id* pairs [2] as described in Section 3 while the second job J_2, which is executed after J_1 has been completed, distributes almost equally these pairs to the R_js, which perform the corresponding distance computations. The M_is of J_1 create a $<K_1, V_1>$ tuple from each record, where K_1 is the blocking key value and V_1 is the value of the *Id* field of that record. All tuples associated with a certain value of K_1 are forwarded to a single R_j [3], by using a simple hash mechanism which will be explained in Section 5. Then, each R_j formulates *Id* pairs by using those tuples which belong to different data sets. The output of each R_j of J_1, which is a volume of *Id* pairs each denoted by W_i ($w_i = |W_i|$), is used

[2]We use the terms *Id* pairs and pairs interchangeably.
[3]This actually means that the records represented by these tuples are inserted into the same block.

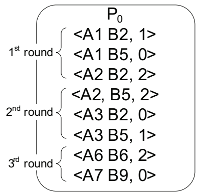


Figure 1: A partitioner task of J_2 performing 3 allocation rounds of distance computations to the reduce tasks.

Algorithm 1 Executed for each *Id* pair by each M_i of J_2

```
Input: $c$, $S$, $K_2$
Output: $<K_2, V_2>$
 1: while ($true$) do
 2:     $V_2 \leftarrow random([0, N_R))$;
 3:     if ($V_2 \notin S$) then
 4:         $S \leftarrow S \cup V_2$;
 5:         break;
 6:     end if
 7: end while
 8: $emit(K_2, V_2)$;
 9: $c \leftarrow c + 1$;
10: if ($c == N_R$) then
11:     $c \leftarrow 0$;
12:     $S \leftarrow \emptyset$;
13: end if
```

as the input to the M_is of J_2. Each such M_i for each pair of W_i formulates a $<K_2, V_2>$ tuple where K_2 holds the pair of *Id* values and V_2 represents the index of the R_j to which this pair will be forwarded. The choice for the value of V_2 occurs on a random basis in repetitive allocation rounds as follows. For each pair, we pick randomly, uniformly, and without replacement an index from the interval $I = [0, N_R)$. The chosen index represents an available R_j and it is assigned to V_2. When all indexes are exhausted, which implies that all available R_js have been chosen, a new allocation round begins. The corresponding P_is simply forward each *Id* pair to the R_j specified by the value of V_2. Finally, the R_js of J_2 perform the distance computations by using the *Id*s held by K_2 in order to retrieve the corresponding records from the data store.
In each round, by choosing randomly and uniformly the index for each R_j from I without replacement, we actually generate *permutations* using the indexes of the R_js included in I. These permuted indexes guarantee a fair distribution of load for the first q_i pairs where $q_i = \lceil w_i/N_R \rceil$. The remainder load of each W_i, denoted by rW_i and consists of $n_i = w_i \ mod \ N_R$ pairs, is also distributed evenly to the available reduce tasks with high probability as shown theoretically in Section 4.1 and experimentally in Section 5.. These rounds are illustrated in Figure 1 where the 3rd round handles the remainder load. Figure 2 outlines the execution of jobs J_1 and J_2 including all the intermediate phases described before.
Algorithm 1 is executed by each M_i of J_2. An allocation round is implemented by using variables c and S. Both are used to store information about the R_js chosen in an allocation round. More specifically, c is a counter, in the context of each M_i, that stores the number of the R_js chosen thus far and S represents the set of the indexes of those R_js. A

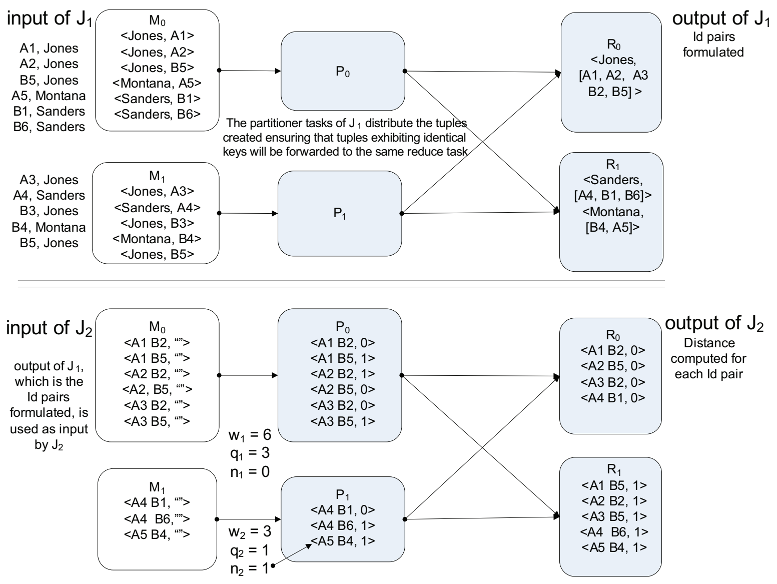


Figure 2: The block of "Jones" includes 5 record pairs which are distributed to the reduce tasks of J_2 almost equally.

randomly chosen index from I is assigned to V_2 provided that this index has not been used before in the current allocation round. As shown in line 10, when c gets equal to the total number of the reduce tasks, a new round of assignments begins by initializing c to 0 (line 11) and S to the empty set (line 12). By doing so, we generate permutations of the indexes of the R_js giving an equal probability to an R_j to be chosen.

4.1 Theoretical analysis

By using the repetitive allocation rounds, described in the previous subsection, we achieve an absolutely equal distribution for the first q_i pairs of each M_i to the R_js. The number of the R_js which will receive the remaining n_i pairs depends on the value of n_i, where $n_i < N_R$.

Theorem 1. *The probability for an R_j to be chosen by an M_i when distributing the remainder load rW_i is:*

$$\Pr[C_{M_i}(R_j)] = \frac{n_i}{N_R},$$

where $C_{M_i}(R_j)$ denotes the corresponding event.

Proof. *During the distribution of each remainder load rW_i, the probability that a certain R_j is not chosen by an M_i in the first choice is $1-(1/N_R)$, by using the complement of $C_{M_i}(R_j)$. In the second choice, it becomes $1-[1/(N_R-1)]$ since we have already used an R_j which is not replaced. Generally, the probability of $C_{M_i}(R_j)$ given n_i pairs is:*

$$\Pr[C_{M_i}(R_j)] = 1-(1-\frac{1}{N_R})(1-\frac{1}{N_R-1})(1-\frac{1}{N_R-2}) \ldots (1-\frac{1}{N_R-n_i+1}) = 1-\frac{N_R-1}{N_R}\frac{N_R-2}{N_R-1}\frac{N_R-3}{N_R-2}\cdots\frac{N_R-n_i}{N_R-n_i+1} = 1-\frac{N_R-n_i}{N_R} = \frac{n_i}{N_R}.$$

□

The above probability increases for large values of n_i, where N_R-1 is the largest, because more chances are given to an R_j to be eventually chosen. On the contrary, if $n_i = 1$, which is the lowest value for n_i, then this probability diminishes. These bounds are illustrated in (1).

$$\frac{1}{N_R} \leq \Pr(C'_{M_i}(R_j)) \leq \frac{N_R-1}{N_R}. \quad (1)$$

Thus, during the distribution of each remainder load rW_i, the R_js are chosen with equal probability, which asymptotically leads to a fair distribution of this load to each R_j.

Corollary 1. *By assuming that the expected value of each n_i is $N_R/2$, it follows that each R_j is chosen $N_M/2$ times in expectation by all map tasks.*

Proof. *Let T_i denote the random variable which holds the number of times an R_j is chosen by all map tasks during the distribution of each rW_i. Then, the expected value of each T_i is:*

$$E[T_i] = \sum_{i=1}^{N_M} \frac{\frac{N_R}{2}}{N_R} = N_M\frac{\frac{N_R}{2}}{N_R} = \frac{N_M}{2}.$$

□

Corollary 2. *The probability for an R_j not to be chosen by any M_i, an event denoted by $C'(R_j)$, during the distribution of each remainder load rW_i is negligible.*

Proof.

$$\Pr[C'(R_j)] = (1-\frac{\frac{N_R}{2}}{N_R})^{N_M} = \frac{1}{2^{N_M}} = o(1).$$

□

5. EVALUATION

We evaluate our method by applying the standard blocking mechanism using data sets extracted from the NCVR [4] list. Each record of these data sets includes 5 fields, namely the *Id*, the *LastName* as the blocking key, the *FirstName*, the *Address*, and the *Town*. The imbalance ratio, which is equal to $\max_{j=0}^{N_R-1} l_j/[\Sigma_{j=0}^{N_R-1}(l_j/N_R)]$, where l_j denotes the number of *Id* pairs forwarded to an R_j, is used to measure the efficiency of each method at reducing the load imbalance among the utilized reduce tasks. The optimal value for the imbalance ratio is 1 because if the maximum load distributed to a reduce task is equal to the average load, then the optimal distribution has been achieved.

We compare our method with the naive random-selection scheme, termed as RND, and with a method introduced in [14], termed as SKETCH, where profiling information of block cardinalities is used. For method RND, we employ both jobs J_1 and J_2 but during the execution of J_2, the selection of the reduce task for each *Id* pair is made randomly and uniformly from I with replacement. Therefore, each reduce task is chosen with probability $1/N_R$. The power of this method is its simplicity in terms of implementation because its only requirement is a simple random function returning integer values from I. Integral components of method SKETCH are specialized data structures, commonly known as sketches, built by fast and small-space algorithms for summarizing massive data in order to provide estimates for cardinality queries. More specifically, the Fast-AGMS sketches [4] are used in order to build profiles of estimated block cardinalities for record pairs which reside on each node of the cluster. These sketches are sent to a single reduce task that combines them and produces a vector where each cell may hold the estimated load for multiple blocks. We evaluate the Cell Block Division algorithm where a division process is applied to those cells which hold large loads in order to divide them into sub-cells before assigning them to the reduce tasks. We set both accuracy parameters ϵ and δ to 0.01. This sketch-based implementation is scalable to massive data sets mainly because it consumes a small amount of memory in order to store the profiling information.

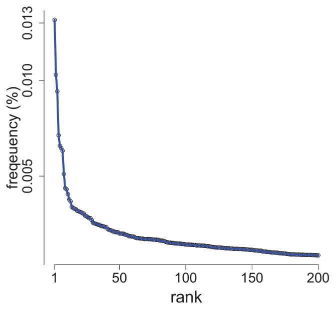
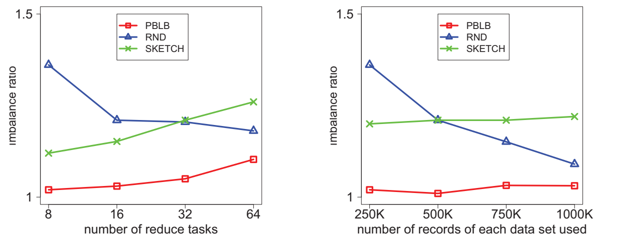


Figure 3: The frequency of the surnames in our data sets indicates a skewed distribution.

Apache Hadoop [5] version 1.0.4 is used as the Map/Reduce implementation running on a cluster of eight compute nodes which are virtual machines of the cloud service of the Greek Research and Academic Community (*Okeanos*) [6]. The software components are developed using the Java programming language version 1.7.

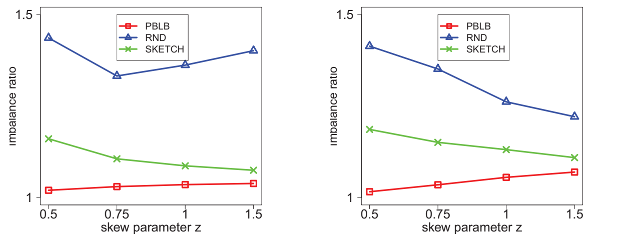
We evaluate the above-mentioned methods by increasing the number of reduce tasks and the size of the data sets at hand performing two series of experiments. In the first series, we compose data sets by extracting records from the NCVR list without inserting any synthetic records. Figure 3 indicates a skewed distribution of surnames where the relative frequency of the first-ranked surname is equal to 0.013 while the frequency of the 200th-ranked surname is equal to 8.6×10^{-4}.



(a) By increasing the number of reduce tasks PBLB exhibits a slight deviation from the optimal value (500K records for each data set).

(b) The performance of PBLB is near-optimal regardless of the size of the data sets used (32 reduce tasks).

Figure 4: Evaluating the performance of each method by using data sets extracted from the NCVR list.



(a) Our method is not affected by the skew applied (32 reduce tasks - 500K records for each data set).

(b) Performance of SKETCH improves as skew increases (64 reduce tasks - 1000K records for each data set).

Figure 5: Evaluating the performance of each method by using synthetic blocks where their cardinality follows the Zipf distribution.

Figure 4a clearly shows that our method does not cause large variations in the distribution of load, exhibiting results very close to the optimal value. Method RND exhibits better results as the number of reduce tasks increases. This happens because the probability of skewing the load, which essentially is the number of collisions exhibited among the chosen reduce tasks, minimizes as the number of reduce tasks increases. Method SKETCH relies on block cardinality estimates which may deviate from the exact values especially

[4] `ftp://www.app.sboe.state.nc.us/data`
[5] urlhttp://hadoop.apache.org/
[6] urlhttps://okeanos.grnet.gr/home/

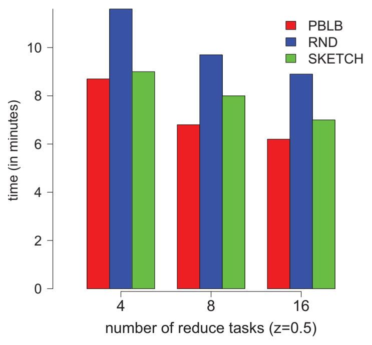


Figure 6: Execution time by increasing the number of reduce tasks ($z = 0.5$).

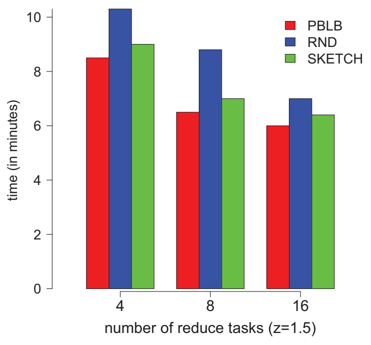
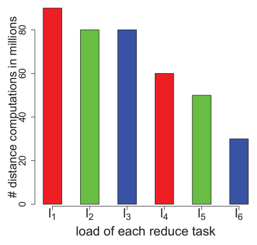


Figure 7: Execution time by increasing both the number of reduce tasks and skew ($z = 1.5$).

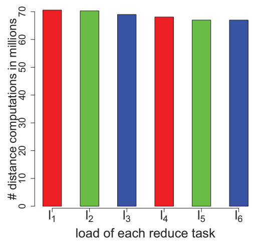
in cases where the skew is not high enough, as reported in [12]. These deviations are reflected to the distribution of load affecting the performance negatively. In Figure 4b, we illustrate experimental results by using 32 reduce tasks and increasing the size of the data sets that participate in the linkage process. The performance of PBLB is very close to the optimal value regardless of the size used. Also, method RND seems to converge to the optimal value (it converges asymptotically) as the size increases.

For the second series of experiments, we generate synthetic blocks by adding record pairs to those blocks generated by the first series of experiments so that their cardinality follows the Zipf distribution. More specifically, the cardinality of the k-th most populated block with record pairs is proportional to k^{-z} where z is the skew parameter. The performance exhibited by our method is not affected by the skew applied, as shown in both Figures 5a and 5b. On the contrary, the performance of method SKETCH improves as the skew increases. This implies the increased accuracy of estimates for block cardinalities provided by the sketches. Method RND is not affected by the skew applied but its performance is worse than the other methods. In Figures 6 and 7, we measure the execution time for all methods by increasing the number of reduce tasks. For these experiments, we used up to 16 reduce tasks because we had 8 virtual machines available and we configured Apache Hadoop to run at most 2 reduce tasks on each machine. Our method exhibits better results as the number of reduce tasks increases without being affected by the skew applied. Increased skew ($z = 1.5$) seems to affect positively the execution time of method SKETCH, as shown in Figure 7, due to more accurate profiling information which results in better utilization of the reduce tasks.

We also ran experiments using the default hash-based P_is used by Apache Hadoop. The map tasks block the records while the reduce tasks perform the distance computations of the record pairs formulated in each block. The default P_is in Apache Hadoop, for each tuple $<K, V>$ map K the index of the reduce task to which this tuple will be forwarded. A hash function of the form $h(K) \mod N_R$ is used where $h(\cdot)$ produces an integer value by K. The imbalance ratio using the data sets from the first series of experiments is constantly above 1.6 and becomes even higher, namely above 1.8, by using the synthetic data sets with skew parameter $z = 1$.



(a) The skew exhibited resulting from the default P_is of Apache Hadoop is quite clear.



(b) PBLB demonstrates an almost equal distribution of load to the R_js.

Figure 8: The number of distance computations performed for the 6 most loaded R_js by using the default hash-based P_is of Apache Hadoop and PBLB.

This imbalance is illustrated in Figure 8a as contrasted to the uniform load distribution achieved by PBLB, which is shown in Figure 8b.

6. CONCLUSIONS

In this paper, we introduce a simple, yet efficient, method for distributing the load of distance computations of record pairs generated by a blocking mechanism on an equal basis among the available reduce tasks by using the Map/Reduce framework. Our method does not require any profiling information but relies on a simple randomization scheme which applies a permutation-based mechanism in order to assign a distance computation to an available reduce task. Further research directions include the mining of statistics regarding characteristics of the considered data sets, as authors do in [9], so that by utilizing a single job, instead of two, we would both formulate and distribute the record pairs to the reduce tasks saving valuable amount of running time.

7. REFERENCES

[1] S. Blanas, J. M. Patel, V. Ercegovac, J. Rao, E. J. Shekita, and Y. Tian. A comparison of join algorithms for log processing in MapReduce. In *Proc. ACM Int. Conf. on Management of Data (SIGMOD)*, pages 975 – 986, 2010.

[2] P. Christen. *Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Springer, Data-Centric Systems and Applications, 2012.

[3] P. Christen. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 24(9):1537–1555, 2012.

[4] G. Cormode and M. Garofalakis. Sketching streams through the net: distributed approximate query tracking. In *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, pages 13 – 24, 2005.

[5] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proc. Int. Conf. on Very Large Databases (VLDB)*, pages 518–529, 1999.

[6] D. Karapiperis and V.S. Verykios. A Distributed Near-Optimal LSH-based Framework for Privacy-Preserving Record Linkage. *Computer Science and Information Systems*, 11(2):745–763, 2014.

[7] L. Kolb, A. Thor, and E. Rahm. Load balancing for mapreduce-based entity resolution. In *Proc. IEEE Int. Conf. on Data Engineering (ICDE)*, pages 618 – 629, 2012.

[8] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia. Skew-Tune: Mitigating Skew in MapReduce Applications. In *Proc. ACM Int. Conf. of Management of Data (SIGMOD)*, pages 25 – 36, 2012.

[9] A. Okcan and M. Riedewald. Processing theta-joins using mapreduce. In *Proc. ACM Int. Conf. on Management of Data (SIGMOD)*, pages 949 – 960, 2011.

[10] A. Rajaraman and J.D. Ullman. *Mining of Massive Datasets*, chapter Finding Similar Items. Cambridge University Press, 2010.

[11] S. Roberts. New York Times: In U.S. Name Count, Garcias Are Catching Up With Joneses. http://www.nytimes.com/2007/11/17/us/17surnames.html, 1997.

[12] F. Rusu and A. Dobra. Statistical analysis of sketch estimators. In *Proc. ACM Int. Conf. of Management of Data (SIGMOD)*, pages 187 – 198. ACM, 2007.

[13] R. Vernica, M. J. Carey, and C. Li. Efficient parallel set-similarity joins using mapreduce. In *Proc. ACM Int. Conf. on Management of Data (SIGMOD)*, pages 495 – 506, 2010.

[14] W. Yan, Y. Xue, and B. Malin. Scalable Load Balancing for MapReduce-based Record Linkage. In *Proc. IEEE Int. Performance Computing and Communications Conf. (IPCCC)*, pages 1 – 10, 2013.