Volume 5, Issue 9, September 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

A Novel MR-based Approach for Blocking of large Scale Patient Records

Amandeep Kaur* CSE & IKGPTU India
Dr. G. N. Verma Applied Science & IKGPTU India
Dr. K. S. Mann Prof. & Dean India

Abstract— Cloud computing has proven to be a powerful ally to efficient parallel execution of data-intensive tasks such as Patient record matching & deduplication (PRMD) in the era of Big Data. For this reason, studies about challenges and possible solutions of how PRMD can benefit from the cloud computing paradigm have become an important demand nowadays. In this paper, we investigate how the MapReduce programming model can be used to perform efficient parallel PRMD using a Adaptive Sorted Neighborhood Method (SNM) that uses a varying size (adaptive) window. We propose MapReduce Duplicate Count Strategy (MR-DCS ++), an efficient MapReduce- based approach for the adaptive SNM, aiming to increase even more the performance of SNM. The evaluation results based on real-world datasets and cloud infrastructure show that our approach increases the performance of MapReduce-based SNM by providing better results for the PRMD execution time.

Keywords— Adaptive, Sorted Neighborhood Method, Patient Record Matching, MapReduce

I. INTRODUCTION

Distributed computing has received a lot of attention lately to perform high data-intensive tasks in the era of Big Data [9]. Extensive powerful distributed hardware and service infrastructures capable of processing millions of these tasks use of such cloud environments, many programming models have being created to deal with a vast amount of data. In this context, MapReduce (MR) [5], a well-known programming model for parallel processing on cloud infrastructures, emerges as a major alternative for the efficient distributed data-intensive tasks due to its capability for being a scalable parallel shared-nothing data-processing.

We investigate the MR-based parallelization for the complex problem of Patient Record Matching (PRMD) (also known as entity resolution, deduplication, record linkage, or reference reconciliation), i.e., the task of identifying patient entities(records) referring to the same real-world object [8]. PRMD is a data-intensive and performance critical task that demands studies on how it can benefit from cloud computing. The task has critical importance for data cleaning and integration [9], e.g., to find duplicate patients or to match demographics on electronic health records.

The PRMD task is a challenging problem nowadays. Besides the need of applying matching techniques on the Cartesian product of all input patient records which leads to a computational cost in the order of O(n2), there is an increasing trend of applications being expected to deal with vast amounts of data that usually do not fit in the main memory of one machine. This means that the application of such approach is ineffective for large datasets. One way to minimize the workload caused by the Cartesian product execution and to maintain the match quality is reducing the search space by applying blocking techniques [9], i.e., to make intelligent guesses of which records have a high probability of representing the same real-world Patient Record. Such techniques work by partitioning the input data into blocks of similar entities and restricting the PRMD process to entities of the same block.

One of the most popular blocking approaches is the Sorted Neighborhood Method (SNM) [10]. It sorts all entities using an appropriate blocking key, e.g., the first three letters of the Patient Record name, and only compares entities within a predefined (and fixed) distance window w. SNM thus reduces the complexity to O(n . w) for the actual matching. Figure 1 shows an execution example of SNM for a window size w = 3. The input set S consists of n = 9 entities (from a to i). All the entities are sorted according to their blocking key K (1, 2, or 3). Initially, the window includes the first three entities (a, d, b) and generates three pairs of comparisons [(a, d), (a, b), (d, b)]. After that, the window is slided down (one Patient Record) to cover the entities d, b, e and two more pairs of comparisons are generated [(d, e), (b, e)]. The sliding process is repeated until the window reaches the last three entities (c, g, i). Note that the number of comparisons generated is (n- w/2) * (w - 1).

However, the SNM presents a critical performance disadvantage due to the fixed and difficult to configure window size: if it is selected too small, some duplicates might be missed. On the other hand, a too large window results in unnecessary comparisons. Note that if effectiveness is more relevant, the ideal window size should be equal to the size of the largest duplicate sequence in the dataset. To overcome this disadvantage, the authors of [6] proposed an efficient SNM variation named as Duplicate Count Strategy (DCS) that follows the idea of increasing the window size in regions of high similarity and decreasing the window size in regions of low similarity. They also proved that their improved variant of DCS, known as DCS++, overcomes the performance of traditional SNM by obtaining at least the same matching results with a significant reduction in the number of Patient Record comparisons.

© 2015, IJARCSSE All Rights Reserved

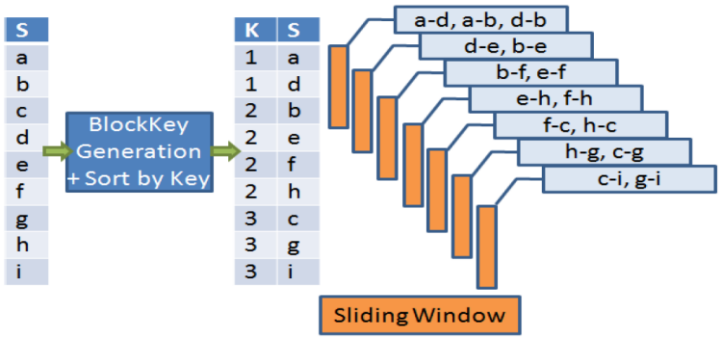

Figure 1: Execution example of the sorted neighborhood method with fixed window size w = 3 (adapted from [10]).

Even with the significant advances in the SNM design, PRMD remains a critical task in terms of performance when applied to large patient records. Thus, this work proposes a MR-based approach capable of combining the efficiency gain achieved by the DCS++ method with the benefit of efficient parallelization of data-intensive tasks in cloud infrastructures to decrease even more the execution time of PRMD tasks performed with the SNM (briefly, combine the best of the two worlds). In this sense, we make the following contributions:

- We propose the MapReduce-based Duplicate Count Strategy (MR-DCS++), a MR-based approach that provides an efficient parallelization of the DCS + + method [6] by using multiple MR jobs and applying a tailored data replication during data redistribution to allow the resizing of the adaptive window. The approach also addresses the data skew problem with an automatic data partitioning strategy that can be combined with MR-DCS++ to provide a satisfactory load balancing across all available nodes.
- We evaluate MR-DCS++ (adaptive window) against the MR-based SNM state of the art approach *RepSN* [10] (fixed window) and show that our approach provides a better performance by diminishing the overall PRMD execution time. The evaluation is performed on a real cloud environment and uses real-world data.

II. BACKGROUND

In this work, we focus on the following PRMD challenge: minimize the execution time necessary to identify similar records in a given dataset. Thinking this way, our optimization goal is: for a given similar identifier operator (matcher) and its inputs, we want to minimize job completion time considering a given amount of available processing nodes. As job completion time, we mean the time spent by all MR jobs. In the remainder of the section, we introduce the MR paradigm, explain how PRMD can be combined with MR and describe the DCS++ method

MapReduce and Patient Record Matching

MapReduce is a programming model designed for parallel data-intensive computing in shared-nothing clusters with a large number of nodes [5]. The main idea of this programming model is to hide details of data distribution and load balancing and let the user focus on the data processing aspects. All the data partitioning and storage is made on a Distributed File System (DFS) and the Entities/records are represented by (key, value) pairs. The computation is expressed with two user defined functions:

map: (keyin , valuein) → *list(keytmp , valuetmp)*

reduce:(keytmp,list(valuetmp))->list(keyout,valueout)

Each of these functions can be executed in parallel on disjoint partitions of the input data. For each input key- value pair, the map function is called and outputs a temporary key-value pair that will be used as input for the reduce function. Unlike the map function, the reduce function is called every time a temporary key occurs as map output. However, within one reduce function only the corresponding values list($value_{tmp}$) of a certain key_{tmp} can be accessed. A MR cluster consists of a set of nodes that run a fixed number of map and reduce jobs. For each MR job execution, the number of map (m) and reduce tasks (r) is specified. The framework-specific scheduling mechanism ensures that after a task has finished, another task is automatically assigned to the released process.

Although there are several frameworks that implement the MapReduce programming model, in the scientific community, Hadoop [1] is the most popular implementation of this paradigm. We therefore implemented and evaluated our approach with Hadoop.

Parallel PRMD implementation using blocking approaches with MR can be done without major difficulties. In a simple way, denoted by *Basic* [9], the map process defines the blocking key for each input Patient Record and outputs a key-value pair (blocking Key, Patient Record). Thereafter, the default hash partitioning in the shuffle phase can use the blocking key to assign the key-value pairs to the proper reduce task. The reduce process will be responsible for performing the Patient Record matching computation for each block.

An evaluation of the *Basic* approach showed a poor performance due to the data skewness caused by varying block sizes [9].This skewness problem can lead to situations in which the execution time may be dominated by a few reduce tasks due to the assignment of the comparisons generated by large windows to a single reduce task. Moreover, since the DCS++ method works with an adaptive window and the reduce tasks must therefore store all entities within the window in main memory (which can lead to serious memory bottlenecks), concerns about load balancing due to data skewness become necessary.

© 2015, IJARCSSE All Rights Reserved

Duplicate Count Strategy (DCS)

The Duplicate Count Strategy (DCS) [6] is based on the SNM and adapts the window size according to the number of already identified duplicate entities. The more duplicates of an Patient Record are found within a window, the larger is the window. On the other hand, if no duplicate of an Patient Record within its neighborhood is found, then DCS assumes that there are no duplicates or the duplicates are very far away in the sorted order of entities. Since the window size increases and decreases according to the number of already identified duplicates, the set of compared entities may be different from the original SNM. Adapting the window size sometimes implies in additional comparisons, but it can also reduce the number of comparisons.

DCS Basic Strategy

The DCS basic strategy consists in increasing the window size by one patient record. Let d be the number of detected duplicates within a window, c the number of comparisons and Øddr a threshold of duplicate detection rate with 0 <Øddr ≤ 1. DCS increases the window size as long as d /c ≥ Øddr. Thus, Øddr defines the average number of detected duplicates per comparison.

DCS++ Strategy

According to the authors of [6], DCS++, a multiple entity increase variant, consists in an improvement of the DCS basic strategy. Instead of increasing the window by just one Patient Record, DCS++ adds for each detected duplicate the next w-1 adjacent entities of that duplicate to the window while d/c < $Ø_{ddr}$. Entities are added only once to that window and the window is no longer increased when d /c < $Ø_{ddr}$ ≤ 1/ w-1 . DCS++ calculates the transitive closure to save some of the comparisons: let us assume that the pairs (t_i, t_k) and (t_i, t_l) are duplicates, with i < k < l. Calculating the transitive closure returns the additional duplicate pair (t_k, t_l). Hence, it is not necessary to verify the window W(k, k + w - 1) and thus this window can be skipped. The key idea used in DCS++ to save unnecessary comparisons is to skip windows (comparisons) by transitive closure. Every time the window slides to the next Patient Record, the size of the window is set to the initial value. An experimental evaluation showing the performance advantages of DCS++ in relation to the traditional SNM is shown in [6].

III. RELATED WORK

Patient Record Matching is a very studied research topic [2]. Many approaches have been proposed and evaluated as described in a recent survey [3]. However, there are only a few approaches that consider parallel Patient Record or Entity matching. The authors of [8] propose a generic model for parallel record matching based on general partitioning strategies that take memory and load balancing requirements into account.

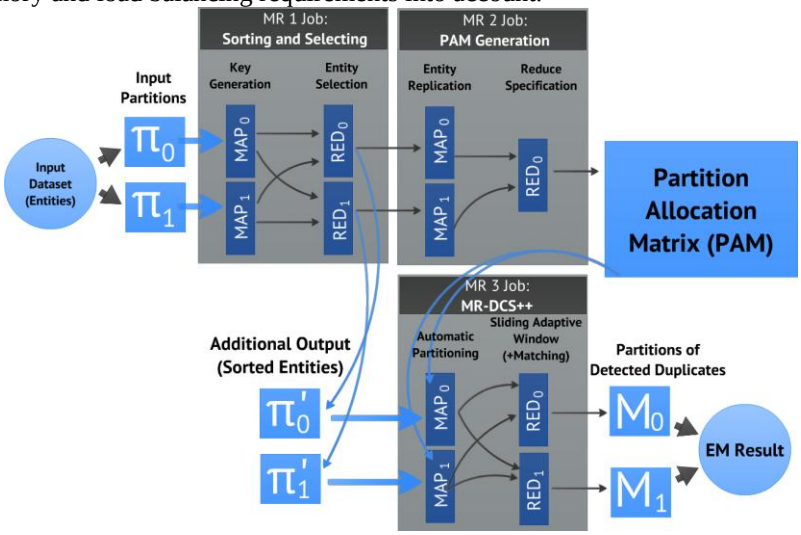

Figure 2: Overview of the MR-DCS++ matching process workflow.

In this context, when we deal with MapReduce-based large- scale Patient Record Matching, two well-known data management problems must be treated: load balancing and skew handling. A few MR-based approaches address the load balancing and skew handling problem. [12] applied a static load balancing mechanism, but it is not suitable due to arbitrary join assumptions. The authors employ a previous analysis phase to determine the data sets' characteristics (using sampling) and thereafter avoid the evaluation of the Cartesian product. This approach focuses on data skew handled in the map process output, which leads to an overhead in the map phase and large amount of map output. MapReduce has already been employed for PRMD (e.g., [14]) but only one mechanism of near duplicate detection by the PPjoin paradigm adapted to the MapReduce framework can be found. [4, 9, 11, 7] study load balancing and skew handling mechanisms to MapReduce-based PRMD for the standard blocking approach. [13] shows another approach for parallel processing Patient Record matching on a cloud infrastructure. This study does not involve the SNM skew handling, but explains how a single token-based string similarity function performs with MR. The approach suffers from load imbalances since some reduce tasks process more comparisons than others.

© 2015, IJARCSSE All Rights Reserved

Finally, the authors of [10] study load balancing for MR- based traditional Sorted Neighborhood Method (SNM). The RepSN approach follows a different blocking approach (fixed window size) that is by design less vulnerable to skewed data. However, its fixed window size design is the reason of the serious disadvantage mentioned earlier in Section I, i.e., if the defined window size is too small, some duplicates might be missed or if a too large window is defined, many unnecessary comparisons will be executed. Due to its proximity with our work, we compared the RepSN approach (state of the art) with our work in an experimental evaluation.

IV. GENERAL MR-BASED DCS++ WORKFLOW

An interesting line of reasoning when we deal with MR- based PRMD is to define an efficient MR approach (with load balancing handling) by knowing previously the Patient Record comparisons generated by the serialized blocking (windowing) method. However, how do we define an efficient MR approach when the blocking (windowing) method adapts according to the duplicate detection rate (like the DCS++ strategy)? How do we assign Patient Record comparisons to the proper reduce tasks with load balancing handling without knowing all the necessary Patient Record comparisons? To answer these research questions, we perform our MR-based DCS++ for PRMD processing using three MR jobs as illustrated in Figure 2. Each MR job is detailed with examples in the following sections.

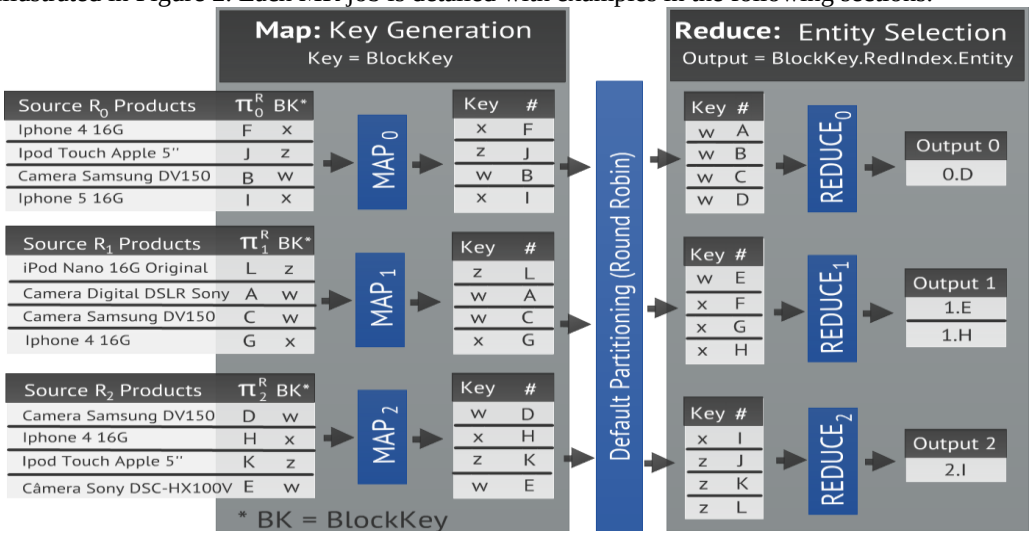

Figure 3: Example dataflow for computation of the Patient Record sorting.

First MR Job: Sorting and Selecting

SNM assumes an ordered list of entities based on their blocking keys. Since the input dataset consists in an unordered Patient Record collection, a preprocessing job is necessary to perform the sorting and to select the boundary entities of the sorted partitions (more details in Section IV). To accomplish this, the map function determines the blocking key for each Patient Record and outputs a key-value pair with a map output key = (blocking key) and value = (Patient Record). The key- value pairs are partitioned based on the average number of entities per reduce task, a = $\sum |\prod| / r$, where $\prod$ represents the set of partitions of Source R and r corresponds to the number of reduce tasks, aiming to avoid the skewness problem and allocate the same number of entities to each reduce task. The reduce task's key-value pairs are sorted by the key (according to SNM) and the reduce function generates the
additional output partitions with the sorted entities. The reduce function also generates an output with the boundary entities of each sorted partition. It selects the last Patient Record of the sorted partition 0, the first and the last entities of intermediate sorted partitions and the first Patient Record of the last sorted partition.

To illustrate, we present a running example with 12 entities (product descriptions) and 3 blocking keys as shown in Figure 3. The example is used throughout Section IV and shows the computation of the sorting and selecting MR job. The 12 entities (A - L) of source R (initially unsorted) are divided (according to the number of map tasks available)
into three input partitions ($\prod$): 0, 1, and 2. Each Patient Record has a blocking key (w, x, and z). The map output key of E is w because E's blocking key equals w. This key is assigned to the second reduce task because the amount of entities assigned to the first reduce task reached the average number of entities per reduce task (a = (4+4+4)/ 3 = 4). The second reduce task generates an additional output sorted partition with the sorted entities E, F, G, and H. An output with the sorted partition's boundary entities E and H preceded by 1 as the corresponding sorted partition index (the second reduce task index) is also generated.

Second MR Job: Partition Allocation Matrix Generation

To understand why the boundary entities of each sorted partition were selected as input of the second MR Job we must recall the concept of boundary pairs. As mentioned earlier, the basic premise of SNM is sorting entities by the
blocking key. Thus, if blocking is done ideally, then the entities outside a block are supposed to be different from the entities within the block. In Figure 3, although entities E and F are neighbors, E belongs to block w and F belongs to block x suggesting that the similarity between E and F is low. However, if the blocking is not ideal, it is quite common that the similarity between entities from different blocks remains considerable. Thus, according to [15], one way to

© 2015, IJARCSSE All Rights Reserved

define the boundaries of these blocks is by establishing a minimum similarity threshold Ømin. In other words, if the similarity between an Patient Record and its next adjacent entity (e.g. E, F) is below Ømin, this pair of entities defines the boundary between two adjacent blocks and is called a boundary pair.

However, DCS++ disregards boundary pairs. It only considers the detection rate to increase or decrease the window. Thus, we show that the use of a Ømin improves the efficiency of DCS++ without changing its duplicate detection rate. The key idea is to show that is unnecessary to increase the window if the similarity between the first and the last Patient Record of the current window is below Ømin. The ideal minimum threshold can be calculated as Ømin = Ømax -α, where Ømax is the maximum threshold that defines if an entity is a duplicate of another one and α is a constant value defined by a data specialist. Thus, this MR job is responsible for indicating the sorted partitions that must be replicated to the proper reduce task in such a manner that the DCS++ adaptive window can be performed without changing the behavior presented by the serialized algorithm. Thus, the output of this MR job is the Partition Allocation Matrix (PAM). The PAM is a m- 1 x m-1 matrix that specifies which sorted partitions must be replicated and attached to other sorted partitions. The PAM computation using MR is straightforward. The map function determines the reduce task that will process each Patient Record and outputs a key-value pair with a composite map output key = (*RedIndex_PartitionIndex*) and the value = (Patient Record). The key-value pairs are partitioned based on the *RedIndex*, i.e., the reduce task index where the Patient Record must be sent, to ensure that the first Patient Record of each partition is assigned to each reduce task whose *RedIndex* < *PartitionIndex*. The last Patient Record of each partition is assigned to the reduce task whose *RedIndex* = *PartitionIndex*. The reduce task's key- value pairs are sorted and grouped by the entire key. The reduce function performs the comparisons between the Patient Record whose *RedIndex* = *PartitionIndex* and each Patient Record *whose RedIndex* <> *PartitionIndex* aiming to find the comparison that returns a similarity value below Ømin. For each comparison performed where the similarity value is above Ømin, the reduce function outputs triples in the form (partition target, partition origin, similarity). The reduce function stops comparing when a similarity value below Ømin is found. Continuing with the running example, in Figure 4, the map function output key of D is 0.0 because D's partition is equal to 0 and D is assigned to the reduce task whose index is 0. This key is assigned to the first reduce task that processes comparisons between D and the first Patient Record of each partition aiming to find the comparison that returns a similarity value below Ømin. Thus, D is compared to E (the first Patient Record of the second partition) and I (the first Patient Record of the third partition). The comparison between D and E returns a similarity value above the Ømin and thus E's partition (1) is replicated and attached to the D's partition (0) to allow the growth of the adaptive window without any possibility of comparison loss. The PAM is updated with the new assignment (PAM[1,0] = 1). Note that the comparison between D and I returns a similarity value below Ømin. This indicates that E's partition contains the boundary pair and thus it is no longer necessary to perform comparisons.

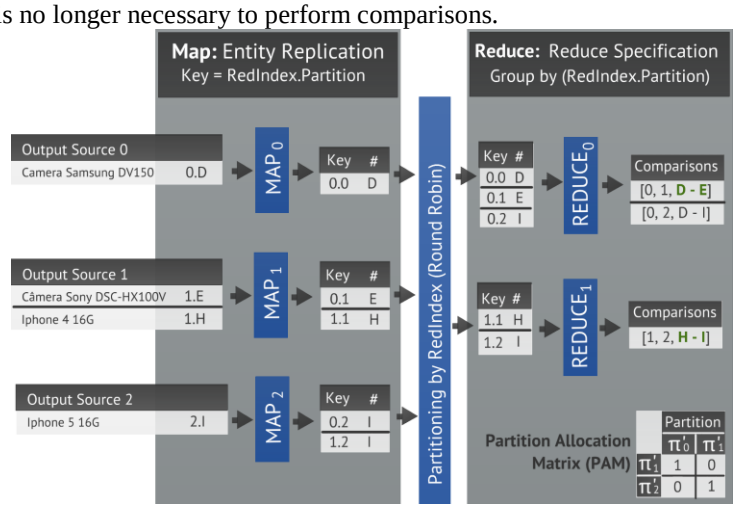

Figure 4: Example dataflow for computation of the Partition Allocation Matrix (PAM).

Third MR Job: MR-DCS++

The third job performs the distributed DCS++ denoted by MR-DCS++. It assigns the sorted partitions to the reduce tasks in such a manner that each reduce function can perform the DCS++ adaptive windowing without missing any relevant comparison. To be more specific, our approach uses the following key ideas:

- MR-DCS++ assigns and attaches sorted partitions to perform the DCS++ adaptive windowing without any loss of relevant comparisons. To prevent the data skewness problem (memory bottlenecks), the sorted partitions are set to have the same number of entities.
- MR-DCS++ improves the load balancing by fixing the same number of entities to each sorted partition. Furthermore, for each reduce task, MR-DCS++ fixes the maximum number of window's slides according to the fixed number of entities in each sorted partition aiming to increase even more the approach's load balancing.

The execution of MR-DCS++ makes use of the PAM as well as the composite map output keys. Each map function generates a well-defined composite key that (together with the associated partition) allows the partition assignments to the proper reduce tasks. The composite key thereby combines information about the target reduce task(s) and the Patient Record itself. The reduce phase performs the PRMD by computing the similarities between entities according to the

© 2015, IJARCSSE All Rights Reserved

DCS++ adaptive windowing strategy. Since the reduce phase consumes the vast majority of the overall runtime (more than 95%, according to our experiments), the time spent due to the overhead caused by the execution of the first two MR jobs and the map phase of the third MR job is negligible. MR-DCS++'s mappers output key-value pairs with key = (*RedIndex* . *PartitionIndex* . *Patient RecordIndex*) and value = (Patient Record). The reduce task index has a value between 0 and r - 1 and is used by the MR function part to perform assignments to the reduce tasks. Since the partition index and the Patient Record index are part of the key and the MR function group takes the entire key into account, it is ensured that each reduce function only receives entities in the correct order. In the map phase, each map m tasks reads the PAM and verifies to which reduce task the entities of its corresponding sorted partitions must be assigned. The number of replications for each Patient Record is defined according to the number of reduce tasks indicated by the PAM. For example, if the PAM indicates that the partition 3 must be assigned to the reduce tasks 1 and 2, the entities belonging to partition 3 must be replicated twice (i.e., to the reduce tasks 1 and 2).

In the reduce phase, each reduce r task receives the assigned entities grouped by the entire key. Since the key has the information of the partition index and the Patient Record index, the entities are placed in the correct order in such a manner that MR-DCS++ can process the adaptive window slide. The size of the first window is passed as a parameter (w). The value is increased according to the specification. However, for load balancing purposes, the window only slides until the last Patient Record whose partition index equals to the reduce task index. Since after every window's slide the window size is set to the initial value, there is no problem on splitting the window sliding among the reduce tasks.

In our running example, as shown in Figure 5, the PAM (generated in the second MR job) indicates that the sorted partition $\prod'_1$ has to be attached to $\prod'_0$ (the process of reading the PAM is by line) and $\prod'_2$ has to be attached to $\prod'_1$ (see Figure 4). The map task replicates the entities of $\prod'_1$ and $\prod'_2$ once. The map task also sets to the *redIndex* of each replicated Patient Record key the same value of the *partitionIndex* indicated by the PAM. For example, E belongs to partition 1 and thus is assigned to reduce task 1, and since the PAM indicates that $\prod'_1$ has to be attached to $\prod'_0$, E is replicated and assigned to the reduce task 0.

Once all entities are positioned, the reduce function starts sliding the window. In our example, we use Ømax = 0.9 (the similarity threshold that indicates if a pair of entities is similar), α = 0.5 (the minimum interval value that the window size can increase and keep performing relevant comparisons) which implies that Ømin = 0.9 – 0.5 = 0.4 (the minimum threshold that indicates if a pair is out of the limits of the boundary pair). The Ømax and α values are defined according to the characteristics of the data. We also use w = 3 which implies that, according to the DCS++ strategy, the increase condition threshold due the duplicate detection rate is Øddr = 1 /w-1 = 1/ 3-1 = 0.5.

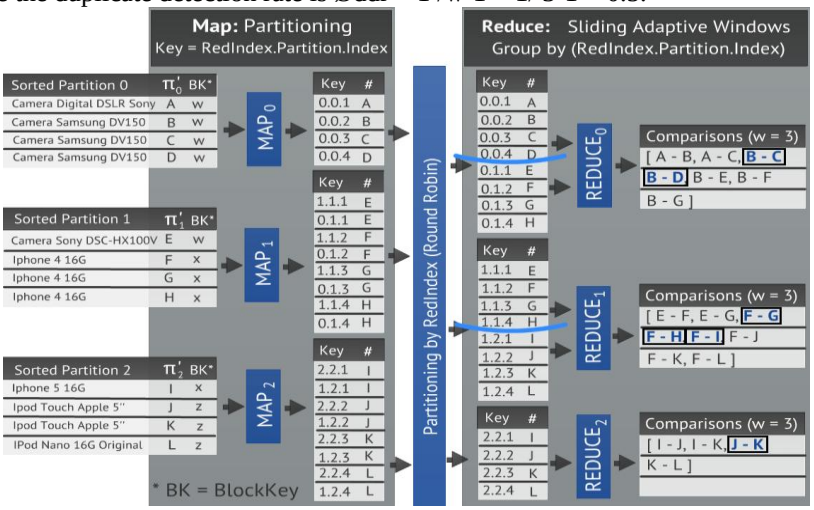

Figure 5: Example dataflow for the MR-DCS++ strategy with w = 3.

Thus, in the reduce task 0, the first window generated covers the entities A, B, and C. This results in the following comparisons: A-B, A-C, and B-C. A-B and A-C are regarded as non-matches whilst B-C is regarded as a match (framed). Since a matching comparison was found, the following relation is tested: if d/ c ≥ 1/w-1, where d is the number of already detected duplicates within the window and c is the number of comparisons already done, then w is increased by $w_{initial}$ -1. Since 1/ 2 ≥ 13-1, w is increased by 2. Now, w covers A, B, C, D and E, the new comparisons generated are B-D and B-E. Since B-D is regarded as a match, the w increase test is loaded again resulting in true (2/ 4 ≥ 0.5) and the window is increased by 2 again. Now, w covers A, B, C, D, E, F and G, B is compared with the rest of the entities within the window. The window is no longer increased due to the lack of new matches. After that, w is set to the initial value (3) and the window slides to the next Patient Record (C). Although the windows starting with C and D present the partition index equal to the reduce index, they are skipped due to the transitive closure related to B (B is similar to C and D). The thick curve in Figure 5 indicates the end of the window's sliding. Also note that the window starting with E is executed in the reduce task 1.

V. CONCLUSION

We propose a novel MR-based approach for solving the problem of the adaptive SNM parallelization, MR-DCS++. The solution provides an efficient parallelization of the DCS+ + method [6] by using multiple MR jobs and applying a

© 2015, IJARCSSE All Rights Reserved

tailored data replication during data redistribution to allow the resizing of the adaptive window. The approach also addresses the data skewness problem with an automatic mechanism of data partitioning that can be combined with MR-DCS++ to ensure a satisfactory load balancing across all available nodes. Our evaluation on a real cloud environment using real-world data demonstrated that MR-DCS++ scales with the number of available nodes. We compared our approach against an existing one (RepSN) and verified that MR-DCS++ overcomes RepSN in performance terms.

In future work, we will investigate how we adapt our solution to address the problem of multiple windows passes. Also, we will investigate how to improve even more the load balancing of our solution and how our approach can be adapted to address other MR-based adaptive techniques for different kinds of data-intensive tasks.

REFERENCES

[1] [1] Apache hadoop. http://hadoop.apache.org/.

[2] [2] P. Christen. Data Matching: Concepts and Techniques for Record Linkage, Patient Record Resolution, and Duplicate Detection. Springer Publishing Company, Incorporated, 2012.

[3] [3] P. Christen. A survey of indexing techniques for scalable record linkage and deduplication. IEEE Trans. on Knowl. and Data Eng., 24(9):1537{1555, Sept. 2012.

[4] [4] G. Dal Bianco, R. Galante, and C. A. Heuser. A fast approach for parallel deduplication on multicore processors. In Proceedings of the 2011 ACM Symposium on Applied Computing, SAC '11, pages 1027{1032, New York, NY, SA, 2011. ACM.

[5] [5] J. Dean and S. GhEMawat. Mapreduce: simplified data processing on large clusters. Commun. ACM, 51(1):107{113, Jan. 2008.

[6] [6] U. Draisbach, F. Naumann, S. Szott, and O. Wonneberg. Adaptive windows for duplicate detection. In proceedings of the 2012 IEEE 28th International Conference on Data Engineering, ICDE '12, pages 1073{1083, Washington, DC, USA, 2012. IEEE Computer Society.

[7] [7] S.-C. Hsueh, M.-Y. Lin, and Y.-C. Chiu. A load-balanced mapreduce algorithm for blocking-based Patient Record-resolution with multiple keys. Parallel and Distributed Computing 2014, page 3, 2014.

[8] [8] T. Kirsten, L. Kolb, M. Hartung, A. Gross, H. Kopcke, and E. Rahm. Data Partitioning for Parallel Patient Record Matching. In 8th International Workshop on Quality in Databases, 2010.

[9] [9] L. Kolb, A. Thor, and E. Rahm. Load balancing for mapreduce-based Patient Record resolution. In Proceedings of

[10] the 2012 IEEE 28th International Conference on Data Engineering, ICDE12, pages 618{629, Washington, DC, USA, 2012. IEEE Computer Society.

[11] [10] L. Kolb, A. Thor, and E. Rahm. Multi-pass sorted neighborhood blocking with mapreduce. Comput. Sc ., 27(1):45{63, Feb. 2012.

[12] [11] D. G. Mestre and C. E. Pires. Improving load balancing for mapreduce-based Patient Record matching. In Proceedings of the Eighteenth IEEE Symposium on Computers and Communications, ISCC'13, pages 618{624. IEEE Computer Society, 2013.

[13] [12] A. Okcan and M. Riedewald. Processing theta-joins using mapreduce. In Proceedings of the 2011 ACM SIGMOD International Conference on ManagEMent of data, SIGMOD '11, pages 949{960, New York, NY, USA, 2011. ACM.

[14] [13] R. Vernica, M. J. Carey, and C. Li. Efficient parallel set-similarity joins using mapreduce. In Proceedings of the 2010 ACM SIGMOD International Conference on ManagEMent of data, SIGMOD '10, pages 495{506, New York, NY, USA, 2010. ACM.

[15] [14] C. Wang, J. Wang, X. Lin, W. Wang, H. Wang, H. Li, W. Tian, J. Xu, and R. Li. Mapdupreducer: detecting near duplicates over massive datasets. In Proceedings of the 2010 ACM SIGMOD International Conference on ManagEMent of data, SIGMOD '10, pages 1119{1122, New York, NY, USA, 2010. ACM.

[16] [15] S. Yan, D. Lee, M.-Y. Kan, and L. C. Giles. Adaptive sorted neighborhood methods for efficient record linkage. In Proceedings of the 7th ACM/IEEE-CS joint conference on Digital libraries, pages 185{194. ACM, 2007.

© 2015, IJARCSSE All Rights Reserved