An Adaptive Blocking Approach for Entity Matching with MapReduce

Demetrio Gomes Mestre[1], **Prof. Dr. Carlos Eduardo Pires**[1]

[1]Programa de Pós-Graduação em Ciência da Computação
Universidade Federal do Campina Grande (UFCG)
Caixa Postal 10.106 – 58.429-900 – Campina Grande – PB – Brazil

demetriogm@gmail.com, cesp@dsc.ufcg.edu.br

Nível: Mestrado
Ano de ingresso no programa: 2012
Defesa da proposta: Dezembro de 2012
Época esperada de conclusão: Março de 2014

Abstract. *Cloud computing has proven to be a powerful ally to efficient parallel execution of data-intensive tasks such as Entity Matching (EM) in the era of Big Data. For this reason, studies about challenges and possible solutions of how EM can benefit from the cloud computing paradigm have become an important demand nowadays. In this context, we investigate how the MapReduce programming model can be used to perform efficient parallel EM using a variation of the Sorted Neighborhood Method (SNM) that uses a varying size window. We propose Distributed Duplicate Count Strategy (DDCS), an efficient MapReduce-based approach for this adaptive SNM, aiming to decrease even more the execution time of SNM.*

Keywords: *MapReduce, Entity Matching, Adaptive Window, Sorted Neighborhood Method.*

1. Introduction

Distributed computing has received a lot of attention lately to perform high data-intensive tasks. Extensive powerful distributed hardware and service infrastructures capable of processing millions of these tasks are available around the world and have being used by industry to streamline its heavy data processing. To make efficient use of these distributed infrastructures, the MapReduce (MR) programming model [Dean and Ghemawat 2008] emerges as a major alternative since it can efficiently perform the distributed data-intensive tasks through map and reduce operations, can scale parallel shared-nothing data-processing and is broadly available in many distributions, such as Hadoop [1].

Entity Matching (EM) (also known as entity resolution, deduplication, or record linkage) is such a data-intensive and performance critical task that demands studies on how it can benefit from cloud computing. EM is applied to determine all entities referring to the same real world object given a set of data sources [Kopcke and Rahm 2010]. For example, in master-data-management applications[2], a system has to identify that the names "Jon S. Stark", "Stark, Jon" and "Jon Snow Stark" are potentially referring to the same person. Thus, the task has critical importance for data cleaning and integration.

Perform EM processes is challenging nowadays. Besides the need of applying matching techniques on the Cartesian product of all input entities which leads to a computational cost in the order of $O(n^2)$, 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. Such techniques work by partitioning the input data into blocks of similar entities and restricting the EM to entities of the same block [Baxter et al. 2003]. For instance, it is sufficient to compare entities of the same manufacturer when matching product offers.

The Sorted Neighborhood Method (SNM) is one of the most popular blocking approaches [Hernández and Stolfo 1995]. It sorts all entities using an appropriate blocking key, e.g., the first three letters of the entity name, and only compares entities within a predefined distance window w. The SNM approach thus reduces the complexity to $O(n \cdot w)$ 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) and 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 $[(a, d), (a, b), (d, b)]$. Later, the window is slided down (one entity) 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 $(n - w/2) \cdot (w - 1)$.

However, the SNM presents a critic 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 many unnecessary comparisons. Note that if effectiveness is most relevant, the ideal window size is equal to the size of

[1] https://hadoop.apache.org

[2] A set of tools that consistently defines and manages the master data (i.e. non-transactional data entities) of an organization.

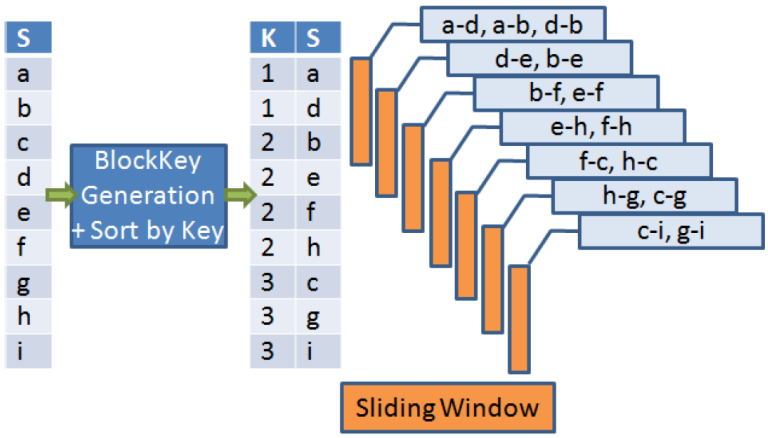

Figure 1. Execution example of the sorted neighborhood method with window size w = 3 (adapted from [Kolb et al. 2012b]).

the largest duplicate sequence in the dataset. Thus, it is not uncommon the intervention of a data specialist to solve this tradeoff (small/large window size). To overcome this disadvantage, the authors of [Draisbach et al. 2012] proposed an efficient SNM variation named as Duplicate Count Strategy or simply **DCS** that follows the idea of increasing the window size in regions of high similarity and decreasing the window size in regions of lower similarity. They also proved that their adaptive SNM overcomes the traditional SNM in performance terms by given at least the same matching results with a significant reduction in the number of comparisons.

Even with significant advances in the SNM design, EM remains a critical task in terms of performance when applied to large datasets. Thus, this work aims to propose a MapReduce-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 EM tasks performed with the SNM (briefly, combine the best of the two worlds).

2. MapReduce and Entity Matching

MapReduce is a programming model designed for parallel data-intensive computing in shared-nothing clusters with a large number of nodes [Dean and Ghemawat 2008]. The key idea relies on data partitioning and storage in a distributed file system (DFS). Entities are represented by (key, value) pairs. The computation is expressed with two user-defined functions:

$$map : (key_{in}, value_{in}) \rightarrow list(key_{tmp}, value_{tmp})$$

$$reduce : (key_{tmp}, list(value_{tmp})) \rightarrow list(key_{out}, value_{out})$$

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 in a shuffle phase to sort the pairs by their keys and send them to 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 tasks (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 programing model, in the scientific community, Hadoop is the most popular implementation of this paradigm. We are therefore implementing and evaluating our approach with Hadoop.

Parallel EM implementation using blocking approaches with MR can be done without major difficulties. In a simple way, denoted as **Basic** [Kolb et al. 2012a], the map process defines the blocking key for each input entity and outputs a key-value pair (blockingKey, entity). 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 is responsible for performing the entity matching computation for each block. An evaluation of the **Basic** approach showed a poor performance due to the data skewness caused by varying size of blocks [Kolb et al. 2012a]. The data skewness problem occurs when the match work of large blocks of entities is assigned to a single reduce task. It can lead to situations in which the execution time may be dominated by a few reduce tasks and thus enable serious memory and load balancing problems when processing too large blocks. Therefore, concerns about lack of memory and load imbalances become necessary.

3. Adaptive Windows for Entity Matching with MapReduce

Given the importance of researching approaches to enhance the Adaptive SNM performance in the context of distributed computing, the goal of this dissertation is to propose an adaptive SNM approach based on the MapReduce model to solve the hard EM task.

To achieve the goal, we divided the work in three phases (with the first two already implemented). In the first phase, we focused on the model development to enable the fully parallelization of the **DCS** method without worrying about the lack of memory or load imbalances in the cloud infrastructure. In the second phase, we extended the model generated in the first phase to address the problem of high memory resources consumption due to the entities replication in the map phase. The third phase is future work and will be discussed in Section 6.

The adaptive SNM MR-based model, named as Distributed Duplicate Count Strategy or simply **DDCS**, proposed in the second phase is defined as the scheme of Figure 2. Two MR processes were used, both based on the corresponding number of tasks and number of input partitions (input dataset). The first process (analysis) consists in a preprocessing step capable of collecting information about the partitions allowed to be replicated and that must be sent to a specific reduce task of the second MR process. This information is formed from the specification of the target reduce task followed by the key index of each partition stored in an array, denoted as Partition Allocation Matrix (PAM), that will be used by the second MR process. The PAM is requested during the execution of the map phase in the second MR process to automatically allocate the partitions to the proper reduce tasks. Furthermore, the PAM enables an efficient redistribution of the entities to the reduce phase, which in turn performs the adaptive sliding window and the entities pairs matching.

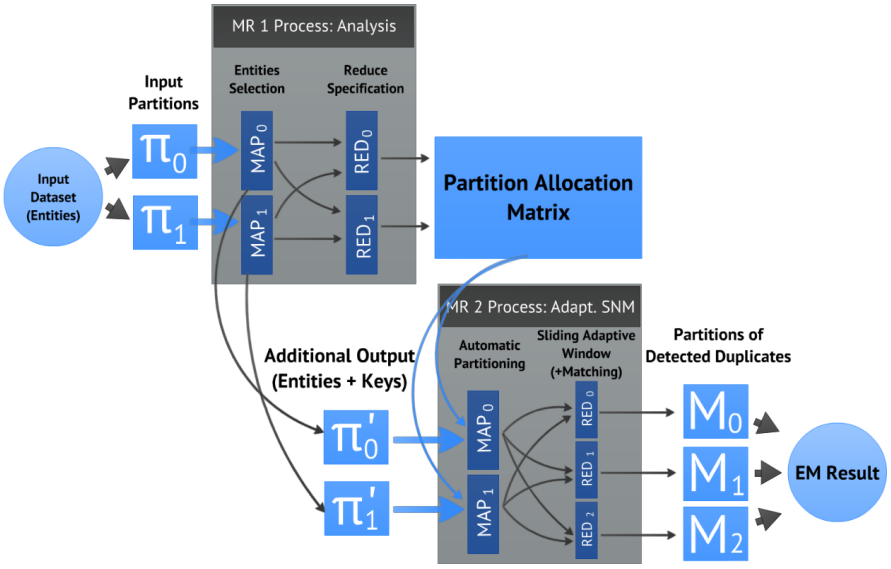

Figure 2. Schematic overview of the Adaptive SNM with MapReduce.

4. Experimental Evaluation

At the present time, we are evaluating **DDCS** against **RepSN** [Kolb et al. 2012b], a MR-based approach for the traditional SNM (fixed window). We are running **RepSN** from the Dedoop[3] aplication provided by the authors of [Kolb et al. 2012b]. The evaluation is regarding two performance-critical factors: the number of configured map and reduce tasks and the number of available nodes in the cloud environment. In each experiment we evaluate the algorithms behavior when dealing with the resources consumption caused by the use of several map and reduce tasks and how they can scale with the number of available nodes.

We are running our experiments on a 10-node HP Pavilion P7-1130 cluster. Each node has one Intel I5 processor with four cores, 4 GB of RAM, and one 1TB of hard disk. Thus the cluster consists of 40 cores and 10 disks. On each node, we installed the Windows 7 64 bits, JAVA 1.6, cygwin and Hadoop 0.20.2. We are utilizing two real-world datasets. The first dataset DS1 is a sample of the Ask[4] database that contains about 214,000 question records. The second dataset DS2, DBLP[5], contains about 1.46 million publication records. For both datasets, the first three letters of the questions or publication title, respectively, form the default blocking key. Since our work focus on performance (execution time), any attribute could have been used to form the default blocking key. Two entities are compared by computing the Jaccard similarity of their comparing attributes and those pairs with a similarity ≥ 0.85 are regarded as matches.

The partial results show that, using DS1 and up to five nodes, although **DDCS** generates 10% more map outputs than **RepSN**, **DDCS** outperforms **RepSN** around 10 to 17% in terms of execution time. Using DS1 and more than five nodes, **DDCS** still generates 10% more map outputs than **RepSN**, but **DDCS** outperforms **RepSN** around 20 to 30% in terms of execution time. The partial result seems to be promising and will be

[3]http://dbs.uni-leipzig.de/howto_dedoop
[4]http://ask.com
[5]http://www.informatik.uni-trier.de/ ley/db/

deeper investigated.

5. Related Work

EM is a very studied research topic. Many approaches have been proposed and evaluated as described in a recent survey [Kopcke and Rahm 2010]. However there are only a few approaches that consider parallel entity matching. The first steps in order to evaluate the parallel Cartesian product of two sources is described in [Kim and Lee 2007]. [Kirsten et al. 2010] proposes a generic model for parallel entity matching based on general partitioning strategies that take memory and load balancing requirements into account.

In this context, when we deal with MapReduce-based large-scale Entity Matching, two well-known data management problems must be treated: load balancing and skew handling. MR has been criticized for having overlooked the skew issue [DeWitt and StoneBraker].

[Okcan and Riedewald 2011] 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 datasets' characteristics (using sampling) and thereafter avoid the evaluation of the Cartesian product. This approach focus 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 EM (e.g., [Wang et al. 2010]) but only one mechanism of near duplicate detection by the PPjoin paradigm adapted to the MapReduce framework can be found. [Kolb et al. 2012a, Mestre and Pires 2013] study load balancing and skew handling mechanisms to MapReduce-based EM for the standard blocking approach. [Vernica et al. 2010] shows another approach for parallel processing entity matching on a cloud infrastructure. This study explains how a single token-based string similarity function performs with MR. This approach suffers from load imbalances because some reduce tasks process more comparisons than the others.

[Kolb et al. 2012b] study load balancing for traditional Sorted Neighborhood Method (SNM). SNM 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 1.

6. Partial Conclusion and Future Work

In this work, we have introduced DDCS, an adaptive Sorted Neighborhood Method based on the MapReduce model that addresses the time-consuming Entity Matching problem. DDCS intends to decrease even more the execution time of the Sorted Neighborhood Method in the cloud by combining a sophisticated strategy of adaptive window with the already renowned MapReduce model. As future work, we will extend DDCS (generated in the second phase, Section 3) to address the load balancing problem (all nodes must have similar working time) by sending an approximate number of comparisons to each node and thus improve the execution time of our method.

References

Baxter, R., Christen, P., and Churches, T. (2003). A comparison of fast blocking methods for record linkage. In *ACM SIGKDD '03 Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, pages 25–27.

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

DeWitt, D. and StoneBraker, M. Mapreduce: A major step backwards, 2008, http://homes.cs.washington.edu/ billhowe/mapreduce_a_major_step_backwards.html.

Draisbach, U., Naumann, F., Szott, S., and Wonneberg, O. (2012). 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. IEEE Computer Society.

Hernández, M. A. and Stolfo, S. J. (1995). The merge/purge problem for large databases. *SIGMOD Rec.*, 24(2):127–138.

Kim, H.-s. and Lee, D. (2007). Parallel linkage. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, CIKM '07, pages 283–292, New York, NY, USA. ACM.

Kirsten, T., Kolb, L., Hartung, M., Gross, A., Köpcke, H., and Rahm, E. (2010). Data partitioning for parallel entity matching. *CoRR*, abs/1006.5309.

Kolb, L., Thor, A., and Rahm, E. (2012a). Load balancing for mapreduce-based entity resolution. In *Proceedings of the 2012 IEEE 28th International Conference on Data Engineering*, ICDE '12, pages 618–629, Washington, DC, USA. IEEE Computer Society.

Kolb, L., Thor, A., and Rahm, E. (2012b). Multi-pass sorted neighborhood blocking with mapreduce. *Comput. Sci.*, 27(1):45–63.

Kopcke, H. and Rahm, E. (2010). Frameworks for entity matching: A comparison. *Data Knowl. Eng.*, 69(2):197–210.

Mestre, D. G. and Pires, C. E. (2013). Improving load balancing for mapreduce-based entity matching. In *Proceedings of the XVIII IEEE symposium on Computers and Communications*, ISCC '13. IEEE Computer Society.

Okcan, A. and Riedewald, M. (2011). 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. ACM.

Vernica, R., Carey, M. J., and Li, C. (2010). 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. ACM.

Wang, C., Wang, J., Lin, X., Wang, W., Wang, H., Li, H., Tian, W., Xu, J., and Li, R. (2010). 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. ACM.