Big Data Integration: Record Linkage

Andrea Giorgi AND.GIORGI4@STUD.UNIROMA3.IT
Computer Science Engineering Department
University of Roma Tre
Registration number 508800

Task: Study and analysis of recent big data integration techniques

Abstract

Record Linkage, also know as Entity Resolution, is a task in which the main goal is to find records in a data set that refer to the same entity. In the big data integration process we can place this task in the middle between Schema Alignment and Data fusion tasks, showing how the necessity of finding the same entity in several data sources, for example in a data lake or in a multiple data storage architecture, it is justified by the enormous quantity of data that modern big data systems has to analyze and process. In this paper we will explore this task and its recent advantages thanks to big data technologies, like Hadoop and MapReduce framework, putting our attention on a load-balancing focused approach.

Keywords: Big Data Integration, Record Linkage, Entity Resolution, Map Reduce

1. Introduction

Record Linkage, or Entity Resolution ER, is the task of disambiguating records that correspond to real world entities across and within data sets. (1) (2). The key aspect of Entity Resolution is about defining what is an entity and which are its different representations across several sources. ER techniques usually are based on a specific pipeline (3), which is composed of three individual steps: [1] Blocking phase, in which small blocks of similar records are efficiently created, [2] Pairwise Matching, which compares all record pairs in a block using pairwise metrics like F1 scores, and [3] Clustering, ensuring semantics. Furthermore as the volume and velocity of data grows the inference across networks and semantic relationships between entities becomes bigger problem. Entity Resolution is able to reduce complexity by proposing standard references to particular entities and deduplicating, lowering the dimensions of the data, or linking entities. This task is facilitated by different programming models, one of them is the MapReduce MR model (4). Effectiveness and scalability of MR implementations depend on effective load balancing approaches to evenly utilize available nodes. The MR model is a good choice for executing blocking-based ER, in which a blocking key is used on the values of one or multiple entity attributes in order to partition the input data into multiple partitions blocks. Blocking-ER in MR is susceptible to severe load imbalances (5), as a consequence large blocks would prevent the utilization of a considerable number of nodes. Using a load balancing approach is possible to use MR for Entity Resolution allowing a faster data integration pipeline.

2. Proposed solution

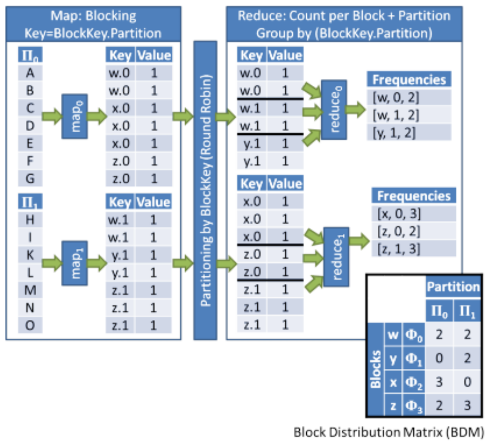
As introduced before the approach proposed here is based on using MapReduce framework and a new load balancing procedure in order to successfully complete an Entity Resolution task. The approach here in study is proposed by Lars Kolb (5) and is based on analysing three different MR solutions for the same problem: [1] a classic one, [2] a block-based load balancing called *BlockSplit* and [3] a pair-based load balancing called *PairRange*. This approach can lead to a significant improvement in execution time, scalability support and cost reduction, thanks to a better management of computing resources.

2.1 Basic MapReduce solution

The input is a set of entities and the output is a match result. There is a generalization on having a valid blocking key, if an entity have not a valid blocking key it needs to be matched with all entities. Parallel ER solution using blocking can be implemented using MapReduce, the map function determines for every entity its blocking key an it gives as result a key-value pair. Using a partition strategy all entities sharing the same blocking key are assigned to the same reduce task. Finally the reducer is called for each block and computes the matching entity pairs within its block. This solution is vulnerable to data skew due to block of variable size and computing time may be dependent largely on one reduce task.

2.2 Block Distribution Matrix BDM

The proposed load balancing strategies are based on two MR jobs, based on the same number of map tasks and the same partitioning of the input data. The first job calculates the Block Distribution Matrix BDM that specifies the number of entities per block separated by input partitions. This matrix is used by the load balancing strategies to tailor entity redistribution for parallel matching of blocks of different size.



		Π0	Π1
w	Φ0	2	2
y	Φ1	0	2
x	Φ2	3	0
z	Φ3	2	3

Figure 1: Block Distribution Matrix and example of use (5)

The BDM is a b x m matrix that specifies the number of entities of b blocks across m input partitions. Using MR the map function determines the blocking key for each entity and outputs a key-value pair with a composite map output key and a corresponding value of 1 for each entity. The key-value pairs are partitioned based on the blocking key component to ensure that all data for a specific block is processed in the same reduce task. The reduce task's key-value pairs are sorted and grouped by the entire key and reduce counts the number of blocking keys per partition and outputs triples of the form ($blockingKey$, $partitionIndex$, $numberOfEntities$).

2.3 BlockSplit approach

BlockSplit processes small blocks within a single match task like a basic MR implementation. Large blocks are spilt according to the m input partitions into m sub-blocks. This sub-blocks are processed using match tasks, each sub-block is processed by the same task. Furthermore pairs of sub-blocks are processed by match tasks in order to evaluate their Cartesian product, ensuring that all comparisons of the original block are computed by the reducer. *BlockSplit* determines the number of comparisons per match task and assigns match tasks in descending size among reduce tasks. This implements a greedy load balancing heuristics that makes unlikely that a large task dominates the overall execution time. *BlockSplit* uses the BDM and a composite map output keys.

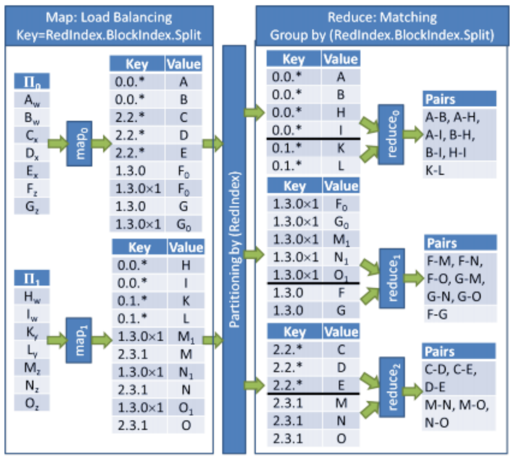

Map: Load Balancing
Key=RedIndex.BlockIndex.Split
Reduce: Matching
Group by (RedIndex.BlockIndex.Split)
Partitioning by (RedIndex)
map₀
map₁
reduce₀
reduce₁
reduce₂
Key
Value
Pairs
A-B, A-H, A-I, B-H, B-I, H-I
K-L
F-M, F-N, F-O, G-M, G-N, G-O
F-G
C-D, C-E, D-E
M-N, M-O, N-O

Figure 2: Dataflow for the BlockSplit approach (5)

The map task outputs key-value pairs with a composite key made starting from $reduceIndex$, $blockIndex$ and *split* value. The reduce task index is a value between [0, r - 1], it is used by partition function to realize the desired assignment to reduce tasks. Thanks to blockIndex each reduce function receives entities of the same block. The split value indicates what match task has to be done by the reduce function. In this approach the map function is responsible of generating the composite key, for each entity if it is not to be partitioned the resulting composite key a classic key-value pair, in the other case the corresponding sub-block is specified. We can notice that the entities of the split blocks are replicated in order to support load balancing and the map function emits the entity as value of the key-value pair; for split blocks we annotate entities with the partition index for use in the reduce phase.

2.4 PairRange approach

BlockSplit strategy splits large blocks according to the input partitions, choiche which in some cases can lead to unbalanced reduce tasks. Another solution can be a *PairRange* strategy, it targets to uniform number of pairs for all reduce tasks. *PairRange* implements a virtual enumeration of all entities and comparisons based on the BDM. The enumeration scheme is used to send entities to one or more reduce tasks and to define the processed pairs. The load balancing strategy is based on splitting the range of all relevant pairs into almost r equally sized pair ranges and assign the k^{th} range R_k to the k^{tk} reduce task. Each map processes its input row by row and can enumerate entities per partition and block. The enumeration of entities allows an enumeration of all pairs to compare in an more efficient way, furthermore pair enumeration employs a column-wise enumeration across all block based on the BDM, Even if entities are processed independently in each partition, the BDM allows to compute the global block-specific entity index locally during the map phase. This approach splits the range of all pairs into almost r equally sized pair ranges, resulting into an assignmnet of the k^{th} range R_k to the k^{tk} reduce task. This assignment leads to assume that k is both the reduce task index and the range index.

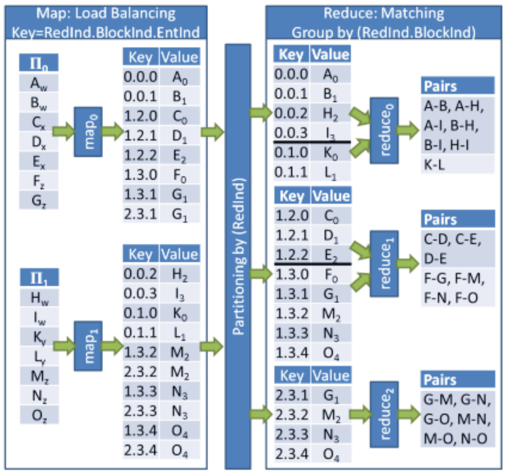


Figure 3: Dataflow for the PairRange approach (5)

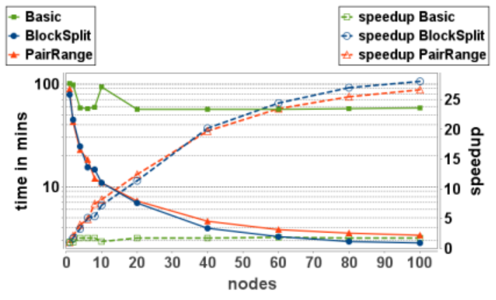
During the initialization each map task reads the BDM and computes the total number of comparison determining the r pair ranges. The map function is now called for rach entity and determines the entity index and all ranges in which the entity is inside, this task can be completed without examining a large number of all pairs thanks to an enumeration scheme which allows for a quick identification of the minimum and maximum index. Finally the map task produces a key-value pair with key $= (rangeIndex, blockIndex, entityIndex)$ and value = entity for each relevant range. The resulting MapReduce partitioning is based on the range index and the sorting is done based on the entire key. As result the reduce task receives only those entities that are relevant for the reduce task's pair range. The reduce function generates all pairs (x, y) with entity indexes $x < y$, checks if the pair index falls into the reduce task's pair range, and in case computes the matching for this pair. To this end, map additionally annotates each entity with its entity index so that the pair index can be easily computed by the reduce function.

3. Implementation

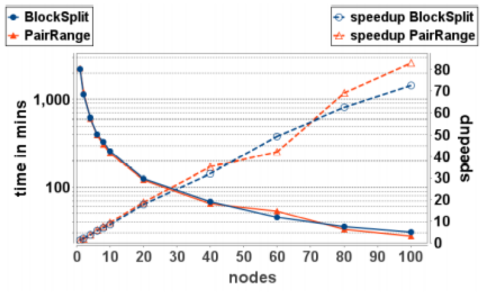
Since MapReduce can be used both locally and in a cloud environment (4), it is possible can choose the latter to take full advantage of the scalability offered by MapReduce and Hadoop. Among the different cloud environments that exist, like Amazon Web Services AWS or Microsoft Azure, a possible implementation could use a service provided by Amazon. EC2 is a web service that provides secure, re-sizable compute capacity in the cloud. It is designed to make web-scale cloud computing easier for developers (6) and its infrastructure fully supports Hadoop and HDFS. Kolb (5) proposes indeed an infrastructure based on EC2 with 100 High-CPU instances with 2 vCore each. Another possible implementation is in a local cluster in which each instance is a Virtual Machine created for the specific job. It seems clear that a solution based on a cloud infrastructure is the most suitable, it brings elasticity and supports scalability as a core feature. A cloud-based ER allows the programmers to execute several different ER analysis, allowing to reuse this new approach on several tasks.

4. Evaluation results

Their evaluation of this new approach is based two different data sets. The second one is bigger than the first one introducing a possible load balancing, execution time and scalability issue since MapReduce is known to present higher execution time comparing to other Big Data technologies like Spark (7). The overall results are promising and allows to conclude that this new ER approach is suitable for various Big Data applications. Talking about execution time it is show that both *BlockSplit* and *PairRange* brings a considerable improvement in computing speed comparing to a classic MapReduce implementation. During the evaluation phase not only the execution time has been taken into account but also scalability, a feature that is important not only for fast computation but also for cost reasons in a cloud environment. The classic approach is unable to scale, but both *BlockSplit* and *PairRange* are able to evenly distribute the workload when new nodes are added to the cluster. This is confirmed by looking the the speedup in computing time, adding new nodes to the cluster brings a speedup up to x25 on the first dataset and a surprising speedup up to 80x on the second dataset, which is bigger, comparing to *Basic* solution. Since their evaluation is done on a cloud environment is difficult to compare it with an evaluation on a local one, but it is possible to assume that even in a local environment, with limited scalability capabilities, the performance increase is still noticeable. MapReduce on a local cluster can suffer of load imbalance and resource shortage because of other applications running on the machine, starting from the very operating system. A load balancing approach can help developers to test their ER applications directly in a local and isolated environment in a more affordable execution time.



(a) Speedup on the first dataset



(b) Speedup on the second dataset

Figure 4: Comparing of execution time for various number of nodes, showing of this two approaches can evenly redistribute their workload (5)

5. Alternative approach

Both presented solutions shows better performances comparing to the classic approach, anyway there should be an automatic choice between both strategy allowing to use each time the most suitable one. Since the characteristics of each method are fixed, before the map phase could be useful to implement a a data set analyzer, which could be able to determine which strategy to use. It is an additional step which could brings more latency to the elaboration and it surely adds up more time to the overall execution time, but using a strategy-advisor each ER task would be completed by the most suitable strategy for that single context. This new approach leads to a complex system which starting from the two different sources *S1* and *S2* a data analysis task is started, determining if the data is prone to be split using as criteria the dimensions of blocks or with pairwise approach. This analysis could be implemented directly on the cloud architecture, having in this case an automated script which retrieves data from our data lake or data warehouse system and performs an automated analysis. This approach could be useful in terms of semantic approach, like finely tuning the parameters of our script in order to keep intact the semantic of data, like hidden relationships that the ER algorithm is not able to compute, or for pruning data which is not perfectly fitting inside our target data schema.

6. Conclusion

MapReduce is an important programming model principally used in Big Data applications which can be used in other scenarios, like Entity Resolution. The solution proposed by Kolb introduces a load balancing technique which brings several improvements to the MR model, starting from a better resource management and node utilization. Both presented approaches show promising results, giving to MR the possibility to be competitive again in terms of data mining and big data integration processes. The big data integration pipeline thanks to this new approaches can be executed rapidly, giving more time to the next step of Data Fusion and saving resources for other operations, like data analysis, which require much more resources for both storage and calculation. Therefore we can say that this MapReduce based approach could be a winning solution in different fields of Big Data.

References

[1] R. B. K. Rossetti, “Basic of entity resolution,” 2021.

[2] A. M. Lise Getoor, “Entity resolution for big data,” 2013.

[3] D. Srivastava, “Big data integration,” 2021.

[4] R. S., S. Kannan, K. Rajakumar, and S. Arumugam, “An overview of the mapreduce model,” 12 2017.

[5] E. R. Lars Kolb, Andreas Thor, “Load balancing for mapreduce-based entity resolution,” 2011.

[6] Amazon, “Amazon elastic compute cloud,” 2021.

[7] N. Samuel, “Hadoop mapreduce vs spark: A comprehensive analysis,” February 2021.