Data Matching and Deduplication Over Big Data Using Hadoop Framework

Pablo Adrián Albanese, Juan M. Ale
palbanese@fi.uba.ar ale@acm.org
Facultad de Ingeniería, UBA

Abstract. Entity Resolution is the process of matching records from more than one database that refer to the same entity. In case of a s ingle database the process is called deduplication. This article proposes a method to solve entity resolution and deduplication problem using MapReduce over Hadoop framework. The proposed method includes data preprocessing, comparison and classification tasks indexing by standard blocking method. Our method can operate with one, two or more datasets and works with semi structured or structured data.

Keywords: entity resolution, data matching, hadoop, mapreduce, standard blocking, indexing

1 Introduction

The Entity Resolution problem, also known as Data Matching or Record Linkage refers to identify, relate and merge records corresponding to the same entity stored in different databases. A special case arises when we analyze duplicates of the same database, this problem is known as duplicate detection. The Entity Resolution process involves the following tasks: 1. Perform data preprocessing, 2. Index records by one or more key fields, 3. Compare records 4. Establish a classification between them (i.e. coincidence, coincidence or not possible match), and 5. Deliver results.

Comparing records is a task that grows exponentially because, for instance, if we have 2 databases of 1,000 records each, the number of comparisons we have to do is 1.000^2 = 1,000,000. In recent decades, the community has developed indexing algorithms order to reduce the number of comparisons.

In the last five years there have been several studies of MapReduce techniques to compare large databases, especially in data matching. These solutions focus in making a better use of parallelism by balancing the records and comparing them using different computational nodes. This problem is called data skew and occurs when the databases contain a non-uniform data distribution which hinders the distribution tasks in different computational nodes used for MapReduce.

In [1] a method is proposed to distribute the load among different computing nodes in order to solve the entity resolution problem with MapReduce using standard blocking.

There are numerous articles where similar methods are proposed. However, most of these articles focus on optimizing the blocking algorithm leaving important questions related to the data matching problem unanswered, such as: How should the preprocessing be included inside MapReduce? Is it possible to use this technique to analyze duplicates of the same database or more than two database? How to perform indexing if we have more than one key field?

This paper extends the proposed method [1] in order to include preprocessing tasks indexed by more than one key field, and record comparison using one, two or more data sets.

The structure of the paper is organized as follows. Related work with an outline of original method is given in Section 2. In Section 3 we present our proposal: a technique to include the preprocessing task and for indexing by more than one key field. We explain where to include matching and classification tasks. Experimental results are presented in Section 4. Conclusions and ideas for future extensions are discussed in Section 5.

2 Related works

In [2] a method for entity resolution on Hadoop that works with semi structured data, including preprocessing tasks, indexing, comparisons and ranking results. It explains how to use different classification techniques and its results to improve future comparisons. The authors in [3] created a method called multi-sig-er, which supports structured and unstructured data type and the tasks contemplate preprocessing of data and reducing comparisons. Another algorithm derived from standard blocking is proposed in [4], however no practical results are provided. Different techniques preprocessing of large amounts of data in MapReduce have been discussed and testing in [5].

The method to extend in this presentation, was presented in [1]. Authors proposed a simple, effective solution to efficiently distribute the calculation of similarity between two entities in entity resolution over MapReduce using two Jobs which the output of the first Job is the input of the second Job. For each Job we must define the Map, Partitioner and Reducer processes. Job 1 is responsible for preparing the information in order for Job 2 can evenly distribute comparisons avoiding the problem of data skew, therefore obtaining a method that is not sensitive to information distribution, so its performance depends only upon the size of the data sets to compare.

3 Our proposal

Many databases contain information with poor quality, incomplete or erroneous information. This problem results in high organizational cost. There are different parameters to measure data quality, the most important are: accuracy, completeness, consistency, timeliness and believability. Accuracy and consistency are the most important in Data Matching. In addition, with Data Matching we face the problem

that data from different databases usually have different formats. When it comes to large amounts of records, we must consider that preprocessing entire databases, consume considerable time and resources. In [5] an experimented of 1 TB data set, preprocessed with MapReduce with 8 nodes, took approximately 63 hours.

For this reason we need a unique method where the data preprocessing be performed in a selectively and efficiently way, because it is not necessary to perform preprocessing to all fields of all records as there are many records that may not need to be compared. The technique proposed consists of reusing the same two Jobs to perform preprocessing task of one field only when needed.

3.1 Distributed Preprocessing Between Job 1 and Job 2

In the proposed technique the task Map of Job 1 processes only key fields, while for the rest of the fields we proceed to apply the acceptance rules. Followed by Job 2 Reducer task where non-key fields are pre-processed in two steps: Step 1. Comparing only against involved fields. Step 2. Comparing against the rest of the fields in case we need to generate that specific record.

We obtain an efficient method that only performs preprocessing tasks as required at each stage of the process.

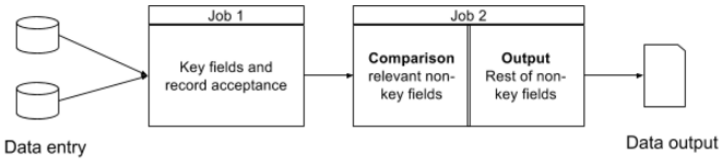


Fig. 1. On map task Job 1 key fields are preprocessed and the necessary fields to define the acceptance criteria of records. In the task reduces the Job 2, the required fields for comparison and classification of records are preprocessed. Finally, only if it is necessary to issue the result, the rest of the fields involved are preprocessed.

3.2 Indexing

The standard blocking method works with a single key field. If we need to search for duplicates by comparing only the records that match on a single field then issues the register in the Map of Job 1. Additionally if we want to find records that match in several key fields, we must generate a common key between them and repeat the same process.

```
function map(key, value)
fileName = get file name from job context
fields = split value
keyField = get key file from fields
idRegister = get id field from fields
write(keyField, fileName + idRegister + fields)
```

The traditional standard blocking method does not contemplate comparing records that match any of the key fields but not necessarily all.

3.3 Indexing by More than One Key Field

We have adapted the task performed by the Map function of Job 1 such that it is able to use more than one key field for comparison of records that may match in any of them:

```
function map(key, value)
fileName = get file name from job context
fields = split value
idRegister = get id field from fields
foreach keyField in fields
    write(keyField, fileName + idRegister + fields)
```

In this new scenario, N pairs (key, value) for each record will be generated, where N is the number of key fields chosen. This modification may generates redundant comparisons of records for cases where two records have similarities in more than one key field. This effect is counterproductive since the performance of the algorithm would begin to depend on the nature of the data and not only the volume.

One way to partially avoid this problem is to use the method Combiner MapReduce. The combiner method runs on the output of the map phase and is used as a filtering or an aggregating step to lessen the number of intermediate keys that are being passed to the reducer. Using this option prevents that Map of Job 2 write duplicate pairs (key, value) that are then being processed by Reducer of Job 2 causing losses in the performance of the process.

```
function combiner(key, values, context)
  value = select first of values
  write(key, value)
```

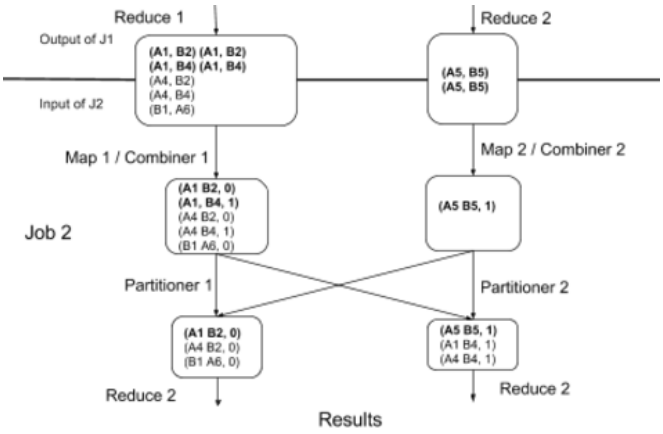


Fig. 2. Pairs of records A1, B2, A1, B4 and A5, B5 have been issued by the Map of Job 1 twice, because they match in two key fields. The Combiner method of Job 2, before being transferred to the reducer of Job 2, groups them into a single entry, preventing comparisons of same records more than once.

As we can see the Map process along with the combiner in Job 2 did not send duplicates to Reduce process values. However this may not always happen, being the combiner just a way to reduce the impact of the records that match in more than one key field.

3.4 Including Matching and Classification

The task of comparing and classifying records, run in the Reducer function of Job 2. As shown in Figure 3, where two records to compare are received, they must obtain the fields in each record, preprocess non-key fields as seen in section 4.2, compare the records, determine the level of similarity between the two records, and if that level is relevant then proceeds to issue the registration.

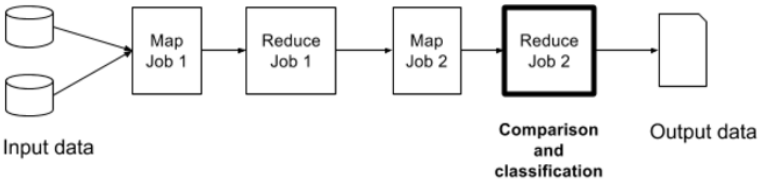


Fig. 3. Comparison and classification are performed in reducer of Job 2.

3.5 Final Method

In this section we summarize the full method when it has 2 data set. Then we explain how to modify this approach for 1 dataset and finally how to apply it when we have more than 1 data set.

1. Data Entry: The allocation of input data set.

2. Preprocessing + exchange key and id fields (Map of Job 1): The preprocessing is performed only in key fields (section 3.1) and for each key field we write pairs key field and id register (section 3.3).
3. Standard partitioner (Partitioner of Job 1): partitioner standard process that distributes tasks in the Reducers available is used.
4. Generation of comparable pairs (Reduce of Job 1): For each key, we write pairs of registers with different file source (entity resolution) or different id (deduplication), pseudocode in section 3.6.
5. Assignment of reducer number (Map of Job 2): Reducer numbers assigned according to the algorithm introduced in [1], the pair (Register 1 Register 2, number of Reducer) is write. A pseudocode is provided:

```
function map(key, value, context)
reducer = 0
reduceAsigned = false
reducers = get number of reducers from context
register1=key
register2=value
while (!reduceAsigned)
do
   randomNumber = get random number(0, reducers)
   if randomNumber not in selectedReducers
      reducer = randomNumber
      add reducer to selectedReducers
end
write (register1 + register2, reducer)
if quantity of selectedReducers = reducers
    selectedReducers = empty
```

6. Combiner (Combiner of Job 2): Verifies that pairs of matching records in more than one key field are not generated more than once (section 3.3).
7. Mapping reducer with reducer number (Reducer of Job 2): sends pairs of records to the reducer indicated by the number of reducer.
8. Comparison + Classification: we proceed to make the necessary preprocessing of non-key fields for the comparison of records and its subsequent classification (section 3.4).

3.6 Application for One Set of Data (deduplication)

To adapt the proposed method to a single database we only need to modify the process 4:

```
function reduce(key, values) // values = registers
processedIds = new list
foreach i values
   foreach j values
     id_i = get id of values[i]
     id_j = get id of values[j]
```

```
      if (id_i <> id_j and id_j not in processedIds)
         write(register_i, register_j)
      Add id_i to processedIds
   End foreach
End foreach
```

3.7 Application for more than Two Databases.

In order to find entities that exist in more than 2 databases, the extended method should be performed N-1 times, where N is the number of data set to be analyzed.

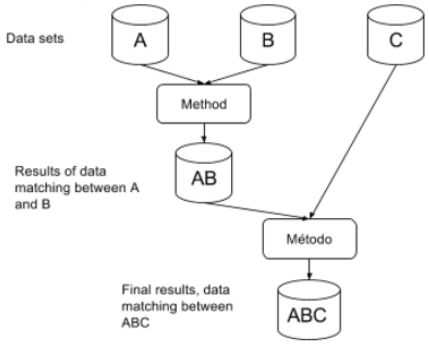


Fig. 3. A, B and C are data sets within which we want to find common entities. First, we run out method between A and B, and then between AB and C .

4 Evaluation

In this section we evaluate the method proposed in the previous section. We use a data set of 250,000 records which keep information about people in a SQL Server database. We generate synthetic records to reach a dataset of 1 million records. Then we divide it into 2 different files.

Table 1. Partial view data set used.

id	first_name	last_name	email_personal	phone_mobile01	country
101	juan carlos	vaccaro	jcvaccaro327@gmail.com		Argentina
102	pablo	salerno	pablos@hotmail.com	+56-1-415449	Chile
103	daniel	ruiz	druiz@yahoo.com	+54-11-12487459	Argentina
105	carolina	Collado	caro_collado@gmail.com	+54-11-25415448	Argentina
110	Magdalena	Gauna	mgauna1979@outlook.com	+54-11-12416446	Argentina

Similar procedures in Transact-SQL language were performed in order to assess that the results are correct. The data set is then exported to two plain text files, to finally upload to an Azure Blob Container which will use our instance of Hadoop for

obtaining and write results. We use Apache Hadoop [1] 2.7.0 with different amounts of nodes (1, 2, 4 and 8) where each node has 4 processors, 7 GB of RAM and 8 hard drives. The provider was Microsoft Azure[2] and Microsoft Blob Service Storage was used to upload the input files.

We use a data set of 500,000 records as the key fields *email_personal* and *phone_personal*. For preprocessing we performed the following task:

1. Disposal of records containing invalid data in either of the two key fields and in the field *firts_name*. (map of job 1)
2. Turn *email_personal* into lowercase (map of job 1)
3. Normalize *phone_personal* to format [country] - [area] - [number] - [internal], if that is not possible, we continue with the formatting of the original field. (map of job 1)
4. Turn *first_name* into lowercase. (reduce of job 2)
5. Encode *first_name* with DobleMethaPhone algorithm. (reduce of job 2)

For record comparison, we consider as a match those records that contain the same *email_person* or *phone_personal* and in both cases the same *first_name*.

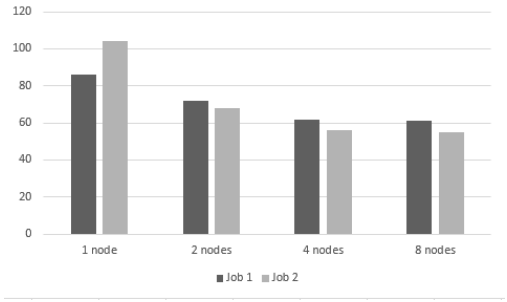
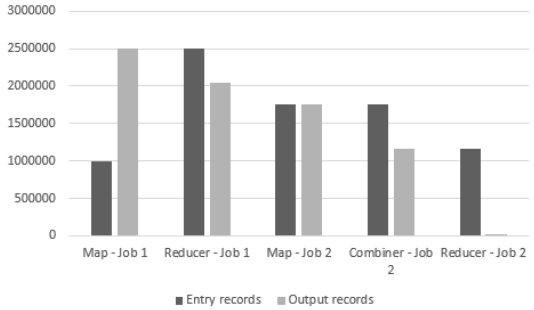


Fig. 4. Execution time according to number of nodes used.

As we increase the number of nodes, runtime decreases, but there is no noticeable difference between the configuration of 4-node and 8-node. This is because it reached its stabilization in 4 nodes.



[1] http://hadoop.apache.org/
[2] https://azure.microsoft.com

Fig. 5. Shows entry and output records processed for each phase. With 1 million input registers finally a little over 1.2 million compare. We can also see how the Combiner avoids comparing 0.5 million records that match the email_personal and phone_personal field.

In figure 5 we can see how the number of records that are transferred between one phase and the other increase in large quantities in relation to the total of records that were entered to compare. Since 1.5 million records (job entry map 1) are entered and finished making 116 million comparisons (entry job reducer 2).

5 Conclusions and Further Research

In this presentation a method for solving entity resolution and deduplication problem using Hadoop Framework is proposed. The proposed method includes preprocessing tasks, indexed by more than one key field, comparison and classification results. We introduce alternatives to work with one and more than 2 data set. The main strengths of our method are: its ease for implementation, its ability to include different preprocessing and classification techniques according to each particular problem, and the possibility of using the method of standard blocking, together with the Combiner of Hadoop, with the use of more than one key field avoiding large number of multiple comparisons of the same records. The proposed solution is scalable and can be used both for structured data as well as for semi-structured data.

Future research could incorporate the use of non-relational and distributed databases such as HBase[3] with the aim of re use the results of comparisons in future comparisons within the same process, as the authors in [2].

References

1. D. Karapiperis and V.S. Verykios. LoadBalancing the Distance Computations in Record Linkage, ACM SIGKDD Explorations Newsletter, Volume 17 Issue 1 (2015)
2. S. Prabhakar Bennya, S. Vasavi, P. Anupriya: Hadoop Framework For Entity Resolution Within High Velocity Streams in CMS 2016, Volume 85, 2016, Pages 550–557 (2016)
3. C. Yan, Y. Song, J. Wang and W. Guo: Eliminating the Redundancy in MapReduce-Based Entity Resolution Cluster, Cloud and Grid Computing (CCGrid)15th IEEE/ACM International Symposium on, Shenzhen, pp. 1233-1236. (2015)
4. Aye Chan Mon, Mie Mie, Su Thwin: Effective Blocking for Combining Multiple Entity Resolution Systems. International Journal of Computer Science Engineering, Vol. 2, No.4**,** pp 126-136 (2013)
5. Qing He, Qing Tan, Xudong Ma, Zhongzhi Shi: The High-Activity Parallel Implementation of Data Preprocessing Based on MapReduce in Lecture Notes in Computer Science, Volume 6401, pp 646-654 (2015)

[3] https://hbase.apache.org/