A Spark-based workflow for probabilistic record linkage of healthcare data *

Robespierre Pita
Distributed Systems Lab.
Federal University of Bahia
Salvador, BA, Brazil
pierre.pita@gmail.com

Clicia Pinto
Distributed Systems Lab.
Federal University of Bahia
Salvador, BA, Brazil
cliciasp1@gmail.com

Pedro Melo
Distributed Systems Lab.
Federal University of Bahia
Salvador, BA, Brazil
pedronovaes@dcc.ufba.br

Malu Silva
Distributed Systems Lab.
Federal University of Bahia
Salvador, BA, Brazil
maludeleon89@gmail.com

Marcos Barreto
Distributed Systems Lab.
Federal University of Bahia
Salvador, BA, Brazil
marcoseb@dcc.ufba.br

Davide Rasella
Institute of Public Health
Federal University of Bahia
Salvador, BA, Brazil
davide.rasella@gmail.com

ABSTRACT

Several areas, such as science, economics, finance, business intelligence, health, and others are exploring big data as a way to produce new information, make better decisions, and move forward their related technologies and systems. Specifically in health, big data represents a challenging problem due to the poor quality of data in some circumstances and the need to retrieve, aggregate, and process a huge amount of data from disparate databases. In this work, we focused on Brazilian Public Health System and on large databases from Ministry of Health and Ministry of Social Development and Hunger Alleviation. We present our Spark-based approach to data processing and probabilistic record linkage of such databases in order to produce very accurate data marts. These data marts are used by statisticians and epidemiologists to assess the effectiveness of conditional cash transfer programs to poor families in respect with the occurrence of some diseases (tuberculosis, leprosy, and AIDS). The case study we made as a proof-of-concept presents a good performance with accurate results. For comparison, we also discuss an OpenMP-based implementation.

Categories and Subject Descriptors

J.1 [**Administrative data processing**]: Government; D.1.3 [**Concurrent Programming**]: Distributed programming.

*This work is partially sponsored by Federal University of Bahia (UFBA), with PIBIC and PRODOC grants, by FAPESB (PPSUS-BA 30/2013 grants), and by CNPq — Brazilian National Research Council.

1. INTRODUCTION

The term big data [18] was coined to represent the large volume of data produced daily by thousands of devices, users, and computer systems. These data should be stored in secure, scalable infrastructures in order to be processed using knowledge discovery and analytics tools. Today, there is a significant number of big data applications covering several areas, such as finance, entertainment, e-government, science, health etc. All these applications require performance, reliability, and accurate results from their underlying execution environments, as well as specific requisites depending on each context.

Healthcare data come from different information systems, disparate databases, and potential applications that need to be combined for diverse purposes, including the aggregation of medical and hospital services, analysis of patients' profile and diseases, assessment of public health policies, monitoring of drug interventions, and so on.

Our work focuses on the Brazilian Public Health System [23], specifically on supporting the assessment of data quality, pre-processing, and linkage of databases provided by the Ministry of Health and the Ministry of Social Development and Hunger Alleviation. The data marts produced by the linkage are used by statisticians and epidemiologists in order to assess the effectiveness of conditional cash transfer programs for poor families in relation to some diseases, such as leprosy, tuberculosis, and AIDS.

We present a four-stage workflow designed to provide the functionalities mentioned above. The second (pre-processing) and third (linkage) stages of our workflow are very data-intensive and time-consuming tasks, so we based our implementation in the Spark scalable execution engine [41] in order to produce very accu-

rate results in a short period of time. The first stage (assessment of data quality) is made through SPSS [17]. The last stage is dedicated to the evaluation of the data marts produced by our pre-processing and linkage algorithms and is realized by statisticians and epidemiologists. Once approved, they load these data marts into SPSS and Stata [6] in order to perform some specific case studies.

We evaluate our workflow by linking three databases: CadÚnico (social and economic data of poor families — approximately 76 million records), PBF (payments from "Bolsa Família" program), and SIH (hospitalization data from the Brazilian Public Health System — 56,059 records). We discuss the results obtained with our Spark-based implementation and also a comparison with an OpenMP-based implementation.

This paper is structured as follows: Section 2 presents the Brazilian Healthcare System to contextualize our work. In Section 3 we discuss some related works specially focusing on record linkage. Our proposed workflow is detailed in Section 4 and its Spark-based implementation is discussed in Section 5. We present results obtained from our case study, both in Spark and OpenMP, in Section 6. Some concluding remarks are presented in Section 7.

2. BRAZILIAN HEALTHCARE SYSTEM

As a strategy to combat poverty, the Brazilian government implemented cash transfer policies for poor families, in order to facilitate their access to education and healthcare, as well as to offer them allowances for consuming goods and services. In particular, the "Bolsa Família" Program [25] was created under the management of the Ministry of Social Development and Hunger Alleviation to support poor families and promote their social inclusion through income transfers.

Socioeconomic information about poor families are kept in a database called CadastroÚnico (CadÚnico) [24]. All families with a monthly income below half the minimum wage per person or a total monthly income of less than three minimum wages can be enrolled in the database. This registration must be renewed every two years in order to keep updated data. All social programs from the federal government should select their recipients based on data contained in CadÚnico.

In order to observe the influence of certain social interventions and their positive (or negative) effects for their beneficiaries, rigourous impact evaluations are required. Individual cohorts [19] have emerged as the primary method for this purpose, supporting the process of improving public policies and social programs in order to qualify the transparency of public investments. It is expected that these transfer programs can positively contribute to the health and the education of beneficiary families, but studies capable to prove this are highly desirable and necessary for the evaluation of public policies.

From an epidemiological standpoint, tuberculosis and leprosy are major public health problems in Brazil, with poverty as one of their main drivers. In addition, there is a broad consensus on the bidirectional relationship between these infectious diseases and poverty: one can lead to another. It is therefore clear that to reduce morbidity and mortality from poverty-related diseases is necessary to plan interventions that address their social determinants.

This work pertains to a project involving the longitudinal study of CadÚnico, PBF ("Bolsa Família" program), and three databases from the Brazilian Public Health System (SUS): SIH (hospitalization), SINAN (notifiable diseases), and SIM (mortality). Table 1 shows these databases with their years of coverage to which we have access. The main goal is to relate individuals in the existing SUS databases with their counterparts in the PBF and CadÚnico, through a process called linkage (or pairing). After linkage, the resulting databases (data marts) are used by statisticians and epidemiologists to analyze the incidence of some diseases in families benefiting from "Bolsa Família" compared to non-beneficiary families.

Databases	Years
SIH (hospitalization)	1998 to 2011
SINAN (notifications)	2000 to 2010
SIM (mortality)	2000 to 2010
CadÚnico (socieconomic data)	2007 to 2013
PBF (payments)	2007 to 2013

Table 1: Brazilan governmental databases.

The major obstacle for linkage is the absence of common identifiers (key attributes) in all databases, which requires the use of probabilistic linkage algorithms, resulting in a significant number of comparisons and in a large execution time. In addition, handling these databases requires the use of secrecy and confidentiality policies for personal information, especially those related to health data. Therefore, techniques for data transformation and anonymisation should be employed before the linkage stage.

The longitudinal study requires the pairing of all available versions for certain databases within the period to be analyzed. In the scope of our project, we must link versions of CadÚnico, PBF, and SIH between 2007 and 2011 to allow a retrospective analysis of the incidence of diseases in poor families and, thereafter, draw up prospects for the coming years. In this scenario, the amount of data to be analyzed, processed, and anonymised tends to increase significantly.

3. CHALLENGES AND RELATED WORK

Record linkage is not a new problem and its classic method was first proposed by [13]. This approach is the basis for most of the models developed later [5]. The basic idea is to use a set of common attributes present in records from different data sources in order to identify true matches.

In [32], probabilistic and deterministic record linkage methods were used to evaluate the impact of the "Bolsa Família" program in education, using some information also contained in CadÚnico. They have proven the importance of database relationships as a tool capable of allowing an integrated view of the information available from various sources, ensuring efficient comparative analysis and increasing the quality and quantity of information required for a search. In public health, many studies use matching records to evaluate impacts or to find patterns [27].

In [11], the authors used probabilistic methods to match records from two SUS databases — SIH (hospitalization) and SIM (mortality) — to identify deaths from ill-defined causes. They developed routines for standardizing variables, blocking based on identification keys, comparison algorithms, and calculation of similarity scores, They also used RecLink [4] to check dubious records for reclassification (as true pairs or not) purposes.

A crucial point is that as the size of databases increases, and therefore the number of comparisons required for record matching, traditional tools for data processing and analysis may not be able to run such applications in a timely manner. In the midst of several studies on software for record linkage, there are few that discuss issues related to the parallelization of processes and data distribution. In [33], some ways to parallelize matching algorithms are discussed, showing good scalability results.

MapReduce paradigm and following technologies have contributed to advance the big data scenario. Some methods to adapt the MapReduce model to deal with record matching are discussed in [16]. Despite these efforts, it is still difficult to find references addressing the problem of matching records using the advantages of MapReduce or similar tools.

Computation techniques related to the preparation steps for record linkage, such as data cleansing and standardization, are still few discussed in the literature. In [31], the authors claim that the cleansing process can represent 75% of the total linkage effort. In fact, preparation steps can directly affect the accuracy of results.

It is possible to observe that management and some aspects of service provision in this context are not yet sufficiently explored [22]. Regarding databases under coordination of public sectors, as CadÚnico and SUS databases, we can observe a high sensitivity and strict requirements for processing and storing such databases in private clusters. Also, there is a lack, mainly in Brazil, of probabilistic matching references over large databases that use the benefits of big data tools.

4. PROPOSED WORKFLOW

Our workflow is divided in four stages, further discussed in the following sections. The first stage corresponds to the analysis of data quality, aiming at to identify, for each database, the attributes more suitable for the probabilistic record linkage process. The set of attributes is chosen based on metrics such as missing values or misfiled records. This step is performed with the support of SPSS software. For security and privacy reasons, the ministries do not allow direct access to their databases; instead, they give us flat files extracted from the databases listed in Table 1. Two people of our team are the ones that manipulate these data based on a strict confidentiality term.

The next stage is pre-processing, being responsible for applying data transformation and cleansing routines in these attributes. We based our implementation on ETL (extract, transform, and load) techniques commonly found in data warehouse tools for standardizing names, filling null/missing fields with default values, and removing duplicate records.

An important step within this stage regards data privacy. We apply a technique based on Bloom filters [34] to anonymize relevant fields prior to the record linkage stage. As stated before, pre-processing is a time-consuming, data-intensive stage, so we use Spark to perform data transformation, cleansing, anonymization, and blocking.

The record linkage stage applies deterministic and probabilistic algorithms to perform pairing. Between CadÚnico and PBF databases, we can use a deterministic algorithm for record linkage based on a common attribute called NIS (social number ID). All beneficiaries of PBF are necessarily registered in CadÚnico and therefore have this attribute. Linkage between CadÚnico and any SUS database (SIH, SINAN, and SIM) must be done through probabilistic algorithms, since there are no common attributes to all databases.

Within SUS databases, the occurrence of incomplete records is quite high, since many records correspond to children or homeless people, which do not always have identification documents or are not directly registered in CadÚnico. In such cases, we try to find a record of an immediate family member, when available. Another very common problem regards incomplete or abbreviated names, which difficults pairing. Again, we use Spark to execute our linkage algorithms in a timely manner and produce the resulting data marts (files with matched and non-matched records).

The last stage is performed by statisticians and epi-

demiologists with the support of statistical tools (Stata and SPSS). The goal is to evaluate the accuracy of the data marts produced by the linkage algorithms, based on data samples from the databases involved. This step is extremely important to validate our implementation and provide some feedback for corrections and adjustments in our workflow.

In the following sections, we discuss a case study on the linkage of CadÚnico and SIH databases made as a proof-of-concept of our workflow. The goal was to generate a data mart covering such databases that is used to analyze the incidence of tuberculosis in PBF beneficiaries and non-beneficiaries families. We chose the databases of the year 2011, respectively with approximately 76 million records and 56,059 records.

4.1 Data Quality Assessment

Attributes suitable for probabilistic matching should be chosen taking into account their coexistence in all databases, their discriminatory capacity, and their quality in terms of filling requirements and constraints. The occurrence of null or missing values is the major problem considered at this stage. This problem can occur by omission or negligence of the operator responsible for filling out forms or by the faulty implementation of the involved information systems. The analysis of missing values is extremely important, because using a variable that has a high incidence of empty fields brings little or no benefit to the matching process.

In our case study, we analyzed the occurrence of null and missing values in the CadÚnico and SIH databases. Tables 2 and 3 show the results obtained for the most significant attributes in each database. Based on the results, we chose three attributes: NOME (person's or patient's name), NASC (date of birtyp), and MUNIC_RES (city of residence).

Attribute	Description	Missing (%)
NIS	Social number ID	0,7
NOME	Person's name	0
DT_NASC	Date of birth	0
MUNIC_RES	City of residence	55,4
SEXO	Gender	0
RG	General ID	48,7
CPF	Individual taxpayer ID	52,1

Table 2: CadÚnico — missing values.

4.2 Data Pre-processing

Datamarts produced in our case study are composed of linked information that reflect the pairing process output. They should contain information about people hospitalized in 2011 with a primary diagnosis of tuberculosis and their socioeconomic data, if registered in CadÚnico, relevant for epidemiological studies.

Attribute	Description	Missing (%)
MUNIC_RES	City of residence	0
NASC	Date of birth	0
SEXO	Gender	0
NOME	Patient's name	0
LOGR	Street name	0,9
NUM_LOGR	House number	16,4
COMPL_LOGR	Address' complement	80,7

Table 3: SIH — missing values.

To facilitate the linkage and increase accuracy, the values of NOME attribute are transformed to uppercase and accents (and possible pontuaction) are removed, so as not to influence the similarity degree between two records. Attributes with null or missing values are treated through a simple substitution to predefined values. This ensures all records are in the same format and contain the same information pattern.

A fundamental concern in our work is confidentiality. We must use privacy policies to guarantee that personal data is protected throughout the workflow. Linkage routines should not be able to identify any person in any database. To accomplish this, we use Bloom filters for record anonymization. Bloom filter is an approach that allows to check if an element belongs to a set. An advantage of this method is that it does not allow false negatives: if a record belongs to the set, the method always returns true. Furthermore, false positives (two records that do not represent the same entity) are allowed. This could be advantageous if the goal is to include records containing small differences in the matched pairs.

The construction of Bloom filters is described in [34] and involves a vector initially populated with 0's. Depending on each attribute, specific positions of this vector, determined by hash functions, are replaced with 1's. Our approach considers an array of 110 positions that maps each bigram (two characters) of the attributes involved. Each attribute affects a fraction of the vector: NOME comprises the first 50 bits, NASC comprises the following 40 bits, and MUNIC_RES the last 20 bits, as shown in Figure 1.

Fractions were chosen considering two aspects: the ideal size a filter must have in order to represent an attribute with a minimum probability error and the influence (or "weight") each field has in the matching decision. Accuracy depends on the filter size (and thus the weight of each attribute), the number of hash functions, and the number of elements added to the filter [36]. The smaller the filter, more errors and false positives are expected because different records can generate very similar vectors with 1's coincidentally mapped in same

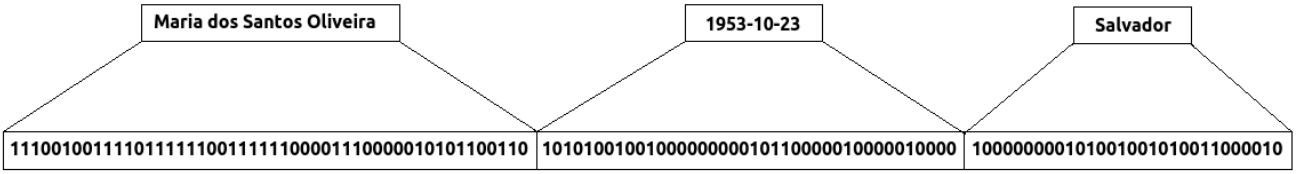

Figure 1: Bit vector generated by Bloom filter.

positions. So, there is a classic tradeoff between size and performance: the vector must be large enough to increase accuracy and, at the same time, small enough to not overload the similarity tests.

For testing our Bloom filter, we constructed three controlled scenarios and use two databases with 50 and 20 records, respectively. The idea was to determine the best vector size and the distribution (number of bits for each field) that provides the best accuracy. Table 4 shows our simulation results. In scenario 3, we simulated filling errors. We can observe that when one attribute has similarity index lower than the expected value, pairing can be saved by the other two attributes that have satisfactory similarity index.

Among all distributions that provide correct results, the distribution with 50, 40, and 20 bits is better for all scenarios. In this sense, the attribute MUNIC_RES must have less influence than NOME because the probability that the same value for NOME in different databases refers to the same person is more significant than two identical values for city.

Another important task performed during the preprocessing stage is blocking construction. The record linkage process requires all records from both databases be compared in order to determine whether they match or not. So, it demands M *x* N comparisons, being M and N the sizes of the databases. However, most of the comparisons will result in non-matched records.

In our case study, the number of comparisons between CadÚnico (approximately 76 million records) and SIH (56,059 records) could be quite prohibitively, so we decide to group records in each database according to a similarity criterion. We chose the MUNIC_RES (city of residence) attribute as blocking key, so that only individuals who live in the same city will be compared. As blocking strategies are a difficult problem, we are also considering another approaches such as adaptative blocking [2], predicates, and phonetic codes (such as Soundex [38], Metaphone [30], and BuscaBR [7]).

4.3 Calculation of Similarity

The decision on pairing two records depends on the analysis of their similarity factor. In this work, we use the Sørensen index [35], also known as Dice [9], to calculate the similarity based on bigrams (two characters) extracted from the bit vector generated by the Bloom filter.

Given a pair of records similar to those shown in Figure 1, the similarity test runs through every bit from both vectors in order to find three metrics: h — representing the count of 1's in the same position in both vectors, a and b — representing the total of 1's in the first and second vectors, respectively, regardless their positions. With these values, it is possible to calculate the Sørensen index using the following formula:

$$D_{a,b} = 2h \ / \ (|a| + |b|)$$

Perfect result expects the number of 1's contained in the first vector added to the second vector be exactly equal to twice the number of common 1's. When this happens, we have a result equal to 1 (great accuracy). If two records turn out different, the value of h decreases and the ratio starts to be smaller than 1.

We use a product by 10,000 to represent the Dice coefficient, therefore values ranging from 0 to 10,000 are used to represent the similarity degree between two vectors. Records are inserted in three distinct groups, as depicted in Figure 2. Every pair whose similarity degree is less than 9,000 is considered non-matched. Values between 9,000 and 9,600 are included in an indecison group for manual analysis, whereas values above 9,600 are considered true matches.

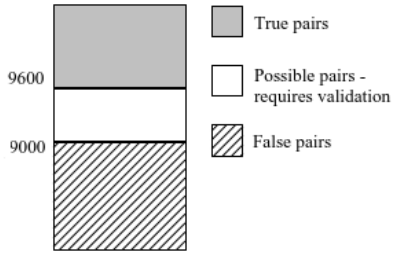

Figure 2: Similarity degrees for Dice calculation.

4.4 Record Linkage

Practical applications of record linkage exist in several areas. For the impact assessment of strategies, for example, it is often necessary to use individual search methods to prove if a specific situation happens in the

Total size and weight distribution	**Scenario 1** No matched records expected		**Scenario 2** Five perfectly matched records expected		**Scenario 3** Expected five matched records with one incorrect character	
	Expected pairings	Pairings found	Expected pairings	Pairings found	Expected pairings	Pairings found
20x20x20	0	310	5	347	5	348
30x30x30	0	29	5	41	5	42
40x40x40	0	11	5	17	5	16
50x50x50	0	0	5	5	5	5
50x50x40	0	0	5	5	5	5
50x40x40	0	0	5	5	5	5
50x40x30	0	0	5	5	5	5
50x30x30	0	2	5	6	5	6
50x40x20	0	0	5	5	5	5

Table 4: Comparison of different vector sizes and weight distributions.

whole group being analysed. Therefore, record linkage is a suitable method to follow cohorts of individuals by monitoring databases that contain continuous outcomes [32]. The interest group can be individually observed in order to obtain more accurate results or to identify variations in the characteristics of each individual. This situation is called a longitudinal study [19].

Probabilistic approaches can be used to match records without common keys from disparate databases. To succeed, we must use a set of attributes for which a probability of pairing can be set. This method requires a careful choice of the keys involved in matching or indeterminancy decisions [10]. This is the case, for example, of determining whether the records "Maria dos Santos Oliveira, Rua Caetano Moura, Salvador" and "Maria S. Oliveira, R. Caetano Moura, Salvador" refer to the same person. The main disadvantages of probabilistic approaches are their long execution times and the debug complexity they impose.

One of the big challenges in probabilistic record linkage is to link records with different schemas and get a good accuracy [11]. There are many problems that hinder pairing, such as abbreviations, different naming conventions, omissions, transcription, and gathering errors. Another big issue is scaling algorithms for large data sets. Transformation and similarity calculation are important challenges for the execution environment when scaled for large databases.

5. SPARK-BASED DESIGN ISSUES

The pioneering programming model capable of handling hundreds or thousands of machines in a cluster, providing fault tolerance, petascale computing, and high abstraction in building applications was MapReduce [8], further popularized by its open-source implementation provided by Hadoop [1]. Basically, this model proposed the division of the input data into splits that must be processed by threads, cores or machines in a cluster responsible for implementing map or reduce functions written by the developer. Intermediate data generated by the first phase are stored on the local disks of processing machines and are accessed remotely by machines performing reduce jobs.

Hadoop was responsible for driving a number of variations seeking to meet specific requirements. Hive [37], Pig [28], GraphLab [20], and Twister [12] are examples of initiatives classified as "beyond Hadoop" [26], which basically keep the MapReduce paradigm but intend to generate new levels of abstractions. However, some authors have indicated significant lacks in MapReduce specially for applications that need to process large volumes of data with strong requirements regarding iterations, machine learning or even with different performance requisites. New frameworks classified as "beyond MapReduce", such as Dremel [21], Jumbo [15], Shark [39], and Spark [41], were created to deal with these new requirements.

Spark is a framework that allows the design of applications based on working sets, the use of some general-purpose languages (such as Java, Scala, and Python), in-memory data processing and a new data distribution model called RDD (resilient distributed dataset) [40]. RDD is a collection of read-only objects partitioned across a set of machines that can be rebuilt if a partition is lost.

The main benefits of using Spark are related to the creation of a RDD for a dataset that must be processed. There are two basic ways to create a RDD, both use the *SparkContext* class: parallelizing a vector of iterable items created at runtime or referencing a dataset in an external storage system (such as a shared filesystem), HDFS [3], HBase [14], or any data source offering a

Hadoop-like InputFormat interface [41].

RDDs can be used through two classes of basic operations: *transformations*, which creates a new dataset from an existing one; and *actions*, which returns a value to the driver program after running a computation on the dataset. The first class is implemented using lazy evaluation and is intended to provide better performance and management of large data sets. Transformations are only computed when an action requires a value to be returned to the driver program [41]. Table 5 shows the main features of Spark framework we used to implement our probabilistic record linkage algorithms.

Transformation	**Meaning**
map(func)	Returns a new RDD by passing each element of the source through *func*
mapPartitions(func)	Similar to map, but runs separately on each partition (block) of the RDD.
Action	**Meaning**
collect()	Returns all the elements of the dataset as an array at the driver program.
count()	Returns the number of elements in the dataset.

Table 5: RDD API used for record linkage.

Another advantage of Spark is its ability to perform tasks in-memory. Views generated during execution are kept in memory, avoiding the storage of intermediate data on hard disks. Spark's developers claim that it is possible to reduce the execution time up to 100 times thanks to the use of working sets, and up to 10 times if hard disks are used. So, our choice to use Spark is justified by its performance, scalability, RDD's fault tolerance, and a very comfortable learning curve due to its compatibility with different programming languages.

The pre-processing stage follows Algorithm 1, which shows how this flow is implemented by the processing of input data transformations using map functions calling other procedures. The intention is that the function *map(blocking)* starts running as *map(normalize)* delivers its results; so we use the *collect()* action to ensure this. It is important to highlight the use of the *cache()* function that fits the memory with the splits extracted from the input files.

Algorithm 1 PreProcessing

```
1: Input ← OriginalDatabase.csv
2: Output ← TreatedDatabaseAnom.bloom
3: InputSparkC ← sc.textFile(Input)
4: NameSize ← 50
5: BirthSize ← 40
6: CitySize ← 20
7: ResultBeta ← InputSparkC.cache().map(normalize)
8: Result ← ResultBeta.cache().map(blocking).collect()
9: for line in Result:
10: write line in Output
11: procedure NORMALIZE(rawLine)
12:     splitedLine ← rawLine.split(;)
13: for fields in splitedLine:
14:     field ← field.normalized(UTF8) return splited-
    Line.join(;)
15: procedure BLOCKING(treatedLine)
16:     splLine ← treatedLine.split(;)
17:     splLine[0] ← applyBloom(splLine[0], NameSize)
18:     splLine[1] ← applyBloom(splLine[1], BirthSize)
19:     splLine[2] ← applyBloom(splLine[2], CitySize)
    return splitedLine.join()
20: procedure APPLYBLOOM(field, vectorSize)
21:     instanceInitialVectorWithSize ← vectorSize
22: for n-grams in field:
23:     bitsVector ← Calculate positions of 1s in Vector
    return bitsVector
```

Algorithm 2 Record linkage

```
1: InputMinor ← TreatedDatabaseAnom1.bloom
2: InputLarger ← TreatedDatabaseAnom2.bloom
3: InputSC1 ← sc.textFile(InputMinor)
4: InputSC2 ← sc.textFile(InputLarger)
5: var ← InputSC1.cache().collect()
6: varbc ← sc.broadcast(var)
7: InterResult ← InputSC2.cache().map(compare)
8: Result ← InputSC2.cache().collect()
9: for line in recordLinkageResult:
10: write line in Output
11: procedure COMPARE(line)
12: for linebc in varbc.value:
13:     get Dice index of (linebc) and (line) comparison
14:     decide about the similarity
15:     if Dice = 9000 then return line
16:     else return None
```

Algorithm 2 shows our record linkage flow. We use a RDD object, since it is read-only, to map the smallest database (SIH). We also use a shared variable, called *broadcast* by Spark, to give every node a copy of the largest database (CadÚnico) in an efficient manner, preventing communication costs, file loads, and split management. A comparsion procedure calculates the Dice index and decides about matching.

6. PERFORMANCE EVALUATION

In order to evaluate the proposed workflow, we ran out our Spark implementation on a cluster with 8 processors Intel Xeon E74820, 16 cores, 126 GB of RAM and a storage machine with up to 10 TB disks connected by the NFS protocol. We compared this implementation with our OpenMP version of the same workflow, also considering other multicore machines: an i5 processor with 4 GB of RAM and 300 GB of hard disk and an i7 processor with 32 GB of RAM and 350 GB of hard disk.

6.1 Spark

For Spark, we chose three samples from CadÚnico and SIH databases, each representing all the cities from the states of Amapá (Sample A), Sergipe (Sample B), and Tocantins (Sample C). These samples represent the smallest Brazilian states in terms of number of records in CadÚnico. Based on them, we can get an idea of the number of comparisons and the rise in the execution time in each case, as shown in Table 6.

Sample Name	Size (in lines) CadÚnico x SIH	Comparisons (millions)	Exec. Time (seconds)
A	367,892 x 147	54,0	96,26
B	1,6 mi x 171	289,5	479
C	1,02 mi x 389	397,63	656,79

Table 6: Spark results for record linkage.

These preliminary results are very promising if we consider the possibility of scaling up the number of machines involved in data processing. Table 7 details the time spent in each stage of the workflow. Standardization, anonymization, and blocking stages are detailed by Algorithm 1 and take only a few minutes in the larger database, while the similarity test and the decision on pairing require a longer running time. The last step consists in recovering a pair of linked records for creating a data mart. Together, all steps do not take more than 12 hours of execution.

	CadÚnico	**SIH**
Size (lines)	approx. 87 mi	approx. 61 k
Standardization	2310.4 s	36.5 s
Anonymization		
Blocking		
Record Linkage	9,03 hours	
Paired Recovery	1,31 hours	

Table 7: Execution time within the workflow.

6.2 OpenMP

The OpenMP interface [29] was chosen due to its syntax simplicity. This kind of implementation divides a task between threads that execute simultaneously, distributed through processors or functional units. The OpenMP API supports C, C++ and Fortran program languages. The C language was chosen because of its worldwide understanding.

The database sets used for this implementation were files containing the results from the Bloom filter applied during the pre-processing stage. These files have N bits in each line (record). The goal is to make the record linkage by calculating the Dice coefficient for each pair of records and writing out the positive Dice results and its respectives lines in an output file.

Access to the database sets in C language is made through pointer types. When a parallel region of the code is initialized, it is necessary to specify global and local variables to the threads. If a pointer is global to the threads, there is a race condition problem if they try to access different positions from the same file. This problem was solved by making these pointers private to each thread. As it is not possible to pass pointer types (only native types), they are created for every line (record) from one of the files. As the files have always the same N bits in each line, it is possible to specify each thread to access uniquely some lines from these files.

Tools	**i5**	**i7**	Cluster
Spark	507.5 s	235.7 s	96.26 s
OpenMP	104.9 s	65.5 s	13.36 s

Table 8: OpenMP x Spark metrics (Sample A).

Table 8 shows the execution time achieved by OpenMP for our Sample A. The machines we used have the following configuration: i5 (4 cores, 8 execution threads), i7 (8 cores, 16 execution threads). The cluster has been described in section 6. The execution time over i7 processor was 37% shorter than i5, showing that the execution time could be even shorter when using computers with more threads per core. Despite its shorter execution time, this approach does not provide a number of advantages offered by Spark, such as scalability and fault tolerance.

The number of matching record was 245. The results were very satisfactory, taking into account that the application runs in only one computer. This shows that the OpenMP implementation is indicated for small linkages or bigger linkages blocked by smaller parts. We are also considering the use of OpenMP for generating the Bloom filter and grouping records to compose the data marts.

7. CONCLUDING REMARKS

The development of a computational infrastructure to support projects focusing on big data from health

systems, like the case study discussed here, was motivated by two factors. First, the need to provide a tool capable of link disparate databases with socioeconomic and healthcare data, serving as a basis for decision-making processes and assessment of the effectiveness of governmental programs. Second, the availability of recent tools for big data processing and analytics, such as those mentioned in this work, with interesting capabilities to deal with new requirements imposed by the applications.

Among the available tools, we chose Spark due to its in-memory facility, its scalability, and ease of programming. Our preliminary tests present very promising results, reinforcing the need for some adjustments in our implementation. New features recently included in Spark could help us, such as the SparkR extension for data quality assessment. We are also testing other techniques throughout the workflow, like phonetic codes, predicates (for blocking) and multi-bit trees.

We plan to continue our tests with OpenMP in order to identify scenarios for which it can provide good performance. The exploration of hybrid architectures (multicore + multi-GPUs) is also in our roadmap.

The execution platform developed in this work represents a major advance in the face of existing solutions for record linkage in Brazil. It will serve as a basis architecture for the installation of a Referral Center for Probabilistic Linkage, and should be supplemented with new features regarding privacy, security, storage, among others.

8. REFERENCES

[1] Apache. Apache Hadoop, 2014. [Online; accessed 12-december-2014].

[2] M. Bilenko, B. Kamath, and R. J. Mooney. Adaptive blocking: learning to scale up record linkage. In *ICDM*, pages 87–96. IEEE Computer Society, 2006.

[3] D. Borthakur. HDFS architecture guide. *Hadoop Apache Project, http://hadoop. apache. org/common/docs/current/hdfs_design.pdf*, 2008.

[4] K. R. Camargo Jr. and C. M. Coeli. RecLink: aplicativo para o relacionamento de bases de dados, implementando o método *probabilistic record linkage*. *Cadernos de Saúde Pública*, 16:439 – 447, 06 2000.

[5] P. Christen. *Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection*. Springer, 1st edition, 2012.

[6] S. Corporation. Stata — data analysis and statistical software, 2014. [Online; accessed 12-december-2014].

[7] F. J. T. de Lucena. Busca fonética em português do Brasil, 2006. [Online; accessed 13-december-2014].

[8] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[9] L. R. Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3), 1945.

[10] A. Doan, A. Halevy, and Z. Ives. *Principles of data integration*. Morgan Kaufmann. Morgan Kaufmann, 2012.

[11] C. L. dos Santos Teixeira, C. H. Klein, K. V. Bloch, and C. M. Coeli. Reclassificação dos grupos de causas prováveis dos óbitos de causa mal definida, com base nas Autorizações de Internação Hospitalar no Sistema Único de Saúde, Estado do Rio de Janeiro, Brasil. *Cad. Saúde Pública*, 22(6):1315–1324, 2006.

[12] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox. Twister: a runtime for iterative MapReduce. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 810–818. ACM, 2010.

[13] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64:1183–1210, 1969.

[14] L. George. *HBase: the definitive guide*. " O'Reilly Media, Inc.", 2011.

[15] S. Groot and M. Kitsuregawa. Jumbo: beyond MapReduce for workload balancing. In *36th International Conference on Very Large Data Bases, Singapore*, 2010.

[16] Y. Huang. Record linkage in an Hadoop environment. Technical report, School of Computing, National University of Singapore, 2011.

[17] IBM. SPSS software — predictive analytics software and solutions, 2014. [Online; accessed 12-december-2014].

[18] IBM, P. Zikopoulos, and C. Eaton. *Understanding big data: analytics for enterprise class Hadoop and streaming data*. McGraw-Hill Osborne Media, 1st edition, 2011.

[19] K. M. Keyes and S. Galea. *Epidemiology matters: a new introduction to methodological foundations*. Oxford University Press, 1st edition, 2014.

[20] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein. Graphlab: a new framework for parallel machine learning. *arXiv preprint arXiv:1006.4990*, 2010.

[21] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis. Dremel: interactive analysis of Web-scale datasets. In *Proc. of the 36th Int'l Conf on Very Large Data Bases*, pages 330–339, 2010.

[22] I. M. Merelli, H. P'erez-Snchez, S. Gesing, and

D. DAgostino. Managing, analysing, and integrating big data in medical bioinformatics: open problems and future perspectives. *BioMed Research International*, 2014(1), 2014.

[23] Ministério da Saúde. Portal da Saúde — Sistema Único de Saúde, 2014. [Online; accessed 12-december-2014].

[24] Ministério do Desenvolvimento Social e Combate à Fome (MDS). Cadastro único para programas sociais do governo federal, 2014. [Online; accessed 12-december-2014].

[25] Ministério do Desenvolvimento Social e Combate à Fome (MDS). Programa Bolsa Família, 2014. [Online; accessed 12-december-2014].

[26] G. Mone. Beyond hadoop. *Communications of the ACM*, 56(1):22–24, 2013.

[27] H. B. Newcombe. *Handbook of record linkage: methods for health and statistical studies, administration, and business*. Oxford University Press, 1st edition, 1988.

[28] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1099–1110. ACM, 2008.

[29] OpenMP.org. The OpenMP API specification for parallel programming, 1998. [Online; accessed 13-december-2014].

[30] L. Philips. Hanging on the Metaphone. *Computer Language*, 7(12), 1990.

[31] S. M. Randall, A. M. Ferrante, and J. Semmens. The effect of data cleaning on record linkage quality. *BMC Medical Informatics and Decision Making*, 13, 06 2013.

[32] J. Romero. Utilizando o relacionamento de bases de dados para avaliação de políticas públicas: uma aplicação para o Programa Bolsa Família. Doutorado, UFMG/Cedeplar, 2008.

[33] W. Santos. Um algoritmo paralelo e eficiente para o problema de pareamento de dados. Mestrado, Universidade Federal de Minas Gerias, 2008.

[34] R. Schnell, T. Bachteler, and J. Reiher. Privacy-preserving record linkage using Bloom filters. *BMC Medical Informatics and Decision Making*, 9:41, 2009.

[35] T. Sørensen. A method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on Danish commons. *Kongelige Danske Videnskabernes Selskab*, 5(4), 1948.

[36] S. Tarkoma, C. E. Rothenberg, and E. Lagerspetz. Theory and practice of Bloom filters for distributed systems. *IEEE Communications Surveys and Tutorials*, 14(1), 2012.

[37] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy. Hive: a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment*, 2(2):1626–1629, 2009.

[38] U.S. National Archives and Records Administration. The soundex indexing system, 2007. [Online; accessed 13-december-2014].

[39] R. S. Xin, J. Rosen, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica. Shark: SQL and rich analytics at scale. In *Proceedings of the 2013 international conference on Management of data*, pages 13–24. ACM, 2013.

[40] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.

[41] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association.