Parallel Entity Resolution with Spark

Author:
Sravani Mantha

November 7, 2016

Advisors:
Dr.-Ing. Eike Schallehn
M.Sc. Xiao Chen

Workgroup Databases & Software Engineering
Mantha, Sravani:
Parallel Entity Resolution with Spark
Abstract

Entity Resolution (ER) is a process of detecting duplicates from records that represent a same entity in the real-world. It is a computation intensive process, which consumes much resource such as time and memory. In recent years, there is a drastic increase in the amount of data being processed and collected. Entity Resolution in such large amounts of data is a complex process, which may take up to several hours and needs large amounts of time and memory to process in a single machine. In order to reduce the runtime of ER and improve its efficiency for large size of data, it can be performed in parallel environments. Current approaches mainly focus on the use of general parallelism or MapReduce-based approaches to develop a parallel ER application in distributed environments. Apache Spark is a new large scale data processing framework, which supports the development of applications in distributed environments and there is not much research on Spark-based ER. However, it supports advanced analytic applications, in-memory operations, built-in libraries and functions. Therefore, in this thesis we implement a parallel ER application using Apache Spark framework and its components. We focus on reducing the runtime of the ER process and also to reach a better effectiveness with learning-based classification methods. To evaluate the application, we use datasets with different sizes. We conducted a series of experiments to comprehensively evaluate our Spark-based ER application.
Acknowledgements

First and foremost, I would like to thank Dr.-Ing. Eike Schallehn for providing me an opportunity to undertake thesis work in his research group.

I would like to express my warmest gratitude to M.Sc. Xiao Chen for her valuable guidance, feedback, and encouragement throughout the thesis work.

I would also like to thank Prof. Myra Spiliopoulou for making time out of her busy schedule for reviewing my thesis.

Finally, I would like to thank my family and friends for encouraging and supporting me all the time.
## Contents

List of Figures .................................................. xii
List of Tables .................................................. xiii
List of Abbreviations .......................................... xv

1 Introduction ................................................... 1

2 Background .................................................... 5
   2.1 Entity Resolution ........................................... 5
      2.1.1 Definition of ER ....................................... 5
      2.1.2 Applications of Entity Resolution ..................... 6
      2.1.3 The Process of Entity Resolution ...................... 7
         2.1.3.1 Pre-processing .................................... 7
         2.1.3.2 Blocking .......................................... 8
         2.1.3.3 Pair-wise Comparison ............................... 11
         2.1.3.4 Classification ..................................... 14
         2.1.3.5 Evaluation ......................................... 17
      2.2 Parallel Entity Resolution ............................... 18
         2.2.1 Parallel Computing ................................... 18
         2.2.2 Parallel ER in a Cluster ............................. 19
         2.2.3 Apache Spark ........................................ 19
      2.3 Summary ................................................ 28

3 Related Work ................................................ 29
   3.1 General Parallelism ....................................... 29
   3.2 MapReduce-based Parallelism .............................. 29
   3.3 Spark-based Parallelism ................................... 30

4 Spark-Based Entity Resolution: Concepts and Implementation 33
   4.1 Motivation ................................................. 33
   4.2 Implementation Choices .................................. 34
      4.2.1 Programming Language ................................. 35
      4.2.2 Development Environments ............................ 35
      4.2.3 External Libraries ................................... 35
4.3 Workflow of Spark-Based Entity Resolution

4.3.1 Pre-processing

4.3.2 Loading data

4.3.3 Blocking

4.3.4 Pair-wise Comparison

4.3.5 Classification

4.3.6 Evaluation

4.4 Tuning

4.5 Summary

5 Evaluation

5.1 Experimental Setup

5.1.1 Input Datasets for Experiments

5.1.2 Experiments Environments

5.1.2.1 Software Environments of the Cluster

5.1.2.2 Hardware Environments of the Cluster

5.1.2.3 Submitting Application to the Cluster

5.2 Experiments Design

5.3 Experiments Results, Evaluation and Discussion

5.3.1 Experiments for Evaluating the Blocking

5.3.2 Experiments for Evaluating Learning-Based Classification

5.3.3 Experiments for Evaluating Speed-up and Efficiency

5.3.4 Experiments for Evaluating Scale-out

5.4 Summary

6 Conclusion and Future Work

6.1 Conclusion

6.2 Future Work

Bibliography
List of Figures

2.1 A simple example of two tables that contain duplicates ................. 6
2.2 Entity Resolution steps [13] ................................................. 8
2.3 Example of decision tree classification .................................. 16
2.4 Example of random forest classification ................................. 16
2.5 Comparison of runtime of logistic regression application between Hadoop and Spark [31] ......................................................... 20
2.6 Code size of Spark compared with other frameworks [23] .......... 20
2.7 Spark unified stack [23] ....................................................... 22
2.8 Comparing runtime of aggregation query using RDD and DataFrame API with various programming languages [3] ......................... 23
2.9 Apache Spark components with DataFrame API ......................... 23
2.10 Workflow of Machine Learning pipeline using DataFrame API ...... 25
2.11 Cluster execution framework in Spark [23] ............................ 26

4.1 Workflow of our ER application ............................................. 36
4.2 Example of records containing duplicates ................................. 37
4.3 Generation of BKVs from name, city and postcode ..................... 38
4.4 The records with the same BKV, combined into one row ............. 38
4.5 Similarity values for attributes calculated using Jaro winkler similarity metrics ................................................................. 39
4.6 Workflow of ML pipeline in our application ............................ 41

5.1 Reduction in the search space for $10^4$ datasets ...................... 51
5.2 Reduction in the search space for $10^5$ datasets ...................... 51
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.3</td>
<td>Reduction in the search space for $10^6$ datasets</td>
<td>51</td>
</tr>
<tr>
<td>5.4</td>
<td>Search space and matches for $10^4$ records</td>
<td>52</td>
</tr>
<tr>
<td>5.5</td>
<td>Search space and matches for $10^5$ records</td>
<td>52</td>
</tr>
<tr>
<td>5.6</td>
<td>Search space and matches for $10^6$ records</td>
<td>52</td>
</tr>
<tr>
<td>5.7</td>
<td>Runtime of ER application with different classification methods</td>
<td>53</td>
</tr>
<tr>
<td>5.8</td>
<td>Precision for $10^4$ records with various classification methods</td>
<td>54</td>
</tr>
<tr>
<td>5.9</td>
<td>Precision for $10^5$ records with various classification methods</td>
<td>54</td>
</tr>
<tr>
<td>5.10</td>
<td>F-measure for $10^4$ records with various classification methods</td>
<td>54</td>
</tr>
<tr>
<td>5.11</td>
<td>F-measure for $10^5$ records with various classification methods</td>
<td>54</td>
</tr>
<tr>
<td>5.12</td>
<td>Runtime of data with $10^6$ records and 10% duplicates with varying amounts of workers using threshold-based classification</td>
<td>55</td>
</tr>
<tr>
<td>5.13</td>
<td>Speed-up of data with $10^6$ records and 10% duplicates</td>
<td>56</td>
</tr>
<tr>
<td>5.14</td>
<td>Efficiency of data with $10^6$ records and 10% duplicates</td>
<td>56</td>
</tr>
<tr>
<td>5.15</td>
<td>Scale-out with varying data and amount of workers</td>
<td>56</td>
</tr>
</tbody>
</table>
List of Tables

2.2 Confusion matrix for evaluating the classified records ................... 17
5.1 Test datasets with various size and amount of duplicates ................ 46
5.2 Configuration of cluster used in experiments ............................... 47
5.3 Experiments design .......................................................... 50
List of Abbreviations

API ........... Application Programming Interface
BKV .......... Blocking Key Value
CPU .......... Central Processing Unit
CSV .......... Comma Separated Values
DAG .......... Direct Acyclic Graph
ER .......... Entity Resolution
ETL .......... Extraction, Transformation and Loading
FN .......... False Negative
FP .......... False Positive
HDFS ........ Hadoop Distributed File System
JDK .......... Java Development Kit
ML .......... Machine Learning
MLlib ........ Machine Learning Library (Spark)
RAM .......... Random Access Memory
RDBMS ...... Relational Database Management System
RDD .......... Resilient Distributed Dataset
RDG .......... Resilient Distributed Graph
SPSS ........ Statistical Package for the Social Sciences
SQL .......... Structured Query Language
SS-Join ...... Set Similarity Join
SSH .......... Secure Shell
TN ............ True Negative
TP ............ True Positive
UI ............ User Interface
VPN .......... Virtual Private Network
XML .......... Extensible Markup Language
YARN .......... Yet Another Resource Negotiator
1. Introduction

In general, it is common that different descriptions of one real-world entity may exist. The different descriptions may be caused due to typographical errors, abbreviations, missing fields, data formatting, etc. These kind of descriptions lowers the data quality and cause wrong interpretation to the user. In order to resolve this problem, Entity Resolution is performed. In the process of Entity Resolution, the records or entities which refer to a same real-world entity are identified. The data in which the Entity Resolution is applied can be taken from single or multiple sources.

Entity Resolution (ER) is a major task in data cleaning and data integration which aim to improve the quality of the data [49]. Over the years, the work related to ER has been attracted researchers from various fields such as bio-informatics, database and data warehousing, data mining, machine learning communities, etc [12]. It is also referred using various names such as data matching, object identification, duplicate detection, etc. It is called as similarity join in database community. The applications of ER can be from various domains such as web pages, census, citation, comparison shopping, public health data, etc [12]. As this data comes from various domains, the format of the data may be structured, semi-structured or unstructured. ER is a complex, computational intensive process which involves many steps in it.

Over the past years, increasing amounts of data are being collected which can be referred to as big data. ER process in large amounts of data generally takes several hours of time for large datasets (big data). To deal with the large amounts of data, the research of ER is directed towards the development of ER process in distributed environments. In distributed environments, the tasks are distributed over several machines, to reduce the runtime and increase the efficiency of the application. So far, various parallel Entity Resolution applications have been developed using new techniques and algorithms. General parallel computing techniques, MapReduce-based algorithms or Spark-based parallelism techniques can be used to perform ER in distributed environments. Compared to general and MapReduce-based parallelism, Spark-based parallelism is new and only
few literature are available. Spark is large scale data processing engine for fast and parallel computing [23]. It was initially started by Matei Zaharia at AMPLab of UC Berkeley in 2009. In 2013, the project was took over by the Apache software foundation under the Apache 2.0 license. Apache Spark is being used for various big data processing applications. Gradually, it became a top level project of Apache due to its promising speed, which is up to 100 times faster than Hadoop MapReduce in-memory and up to 10 times faster in disk. Since existing parallel Entity Resolution framework are MapReduce-based, we aim to implement a parallel ER application using Apache Spark and its components.

**Goal of this Thesis**

The main goal of this thesis is to design and implement a parallel Entity Resolution application using Apache Spark framework. Also, to evaluate the performance of the application using various methods to calculate its effectiveness and efficiency.

To achieve the goal of the thesis, we present the following contributions of our work as below:

- A complete ER application is developed, which involves five different steps of traditional ER application i.e, preprocessing, blocking, pair-wise comparison, classification, and evaluation. All the steps are implemented using Apache Spark framework.

- For maximizing the performance of the application, Spark SQL component has been used which is proven to run faster than a primary abstraction in Spark called Resilient Distributed Datasets (RDD). For classifying the records into matches and non-matches, Spark-based machine learning pipelines are used in our application. Most of the computations in our application are performed in-memory, which helps to run the application faster. Tuning configurations are applied on the application, to improve the parallelism when running in the cluster.

- Datasets with records $10^4$ to $10^6$ are taken which have 10%, 25% and 50% duplicates in each of them to evaluate the performance of the application. To measure the quality of the results, precision, recall and F-measure are calculated. To evaluate the parallelism of the application in the cluster, runtime, speed-up, efficiency and scale-out experiments are conducted.

**Structure of the Thesis**

This document is divided into 6 chapters including the introduction. In the upcoming chapters, we provide detailed information of our parallel ER application. The chapters are structured as follows:

- In Chapter 2, we introduce the fundamental concepts of entity resolution and parallel entity resolution which are necessary to understand the thesis work.
• In Chapter 3, we provide an overview of the related research.

• In Chapter 4, we discuss the concepts and implementation details of our application.

• In Chapter 5, we discuss about the various experiments which are conducted to check the efficiency and effectiveness of our application.

• In Chapter 6, conclusion and future work of the thesis are discussed.
2. Background

In this chapter, we provide fundamental concepts of Entity Resolution (ER) and parallel ER that are required to understand this thesis. In Section 2.1, we present necessary concepts, the steps involved in the process of entity resolution. In Section 2.2, we introduce parallel ER, which aims to solve entity resolution efficiently and scalably with parallel techniques so that ER can face the challenge that the larger and larger data volume has brought.

2.1 Entity Resolution

In this section, we discuss the concepts and techniques of ER. In Section 2.1.1, we explain what ER is. Next, the applications of ER are discussed in Section 2.1.2. At last, in Section 2.1.3 we represent a general process of ER and describe each step in it in detail.

2.1.1 Definition of ER

In the process of ER, records from a single source or multiple sources that represent same real-world entities are identified. The entities that are same in the real-world are called duplicate entities [12]. Generally, duplicates prevail in various domains of data records like census data, health records, geographical data, company records, etc. The input sources of ER can be from databases [2], web pages [16], etc. In the research community, entity resolution is also described using other terms such as duplicate detection [43], deduplication [50], merge/purge problem [28], fuzzy duplicates [2] and record linkage [57]. Particularly, in the database community, ER refers to identifying similar records from databases that symbolize same real-world entities [7]. It would be easy to detect duplicates from a database if they are exactly the same records, by using ‘distinct’ keyword. However, duplicates are caused by abbreviations, typographical errors, misspelled words, missing fields, etc., which makes it hard to identify them using a single query [8].
The input records for ER may be in various forms of data like structured (relational databases), semi-structured (XML) and unstructured data (text). The ER process depends on the format of the data. Therefore, various entity resolution techniques have been proposed for distinct formats of the data [7, 38].

In order to understand ER, let us consider a simple example in Figure 2.1, where there are two tables $A$ and $B$, which contain personal information of the customers related to a company. We can observe that both tables contain same fields: ID, Name, City, and Date of Birth. The representation of the dates in Date of Birth column in table $B$ is different from table $A$. Furthermore, a few entries of personal details contain errors and null values. However, we can see that $A1$, $B1$ refer to the same person named James and $A4$, $B3$ refer to the same person named Victoria. ER is the process to see through those different formats, errors, etc; and to find that the records belong to a same person. From this example, it is very easy to find duplicates manually as there are only several records. But, in the real world, the data that contains duplicates is of large size and more complex, which cannot be identified manually.

2.1.2 Applications of Entity Resolution

In this section, we provide the main applications where Entity Resolution (ER) is applied. Here are typical applications of ER:

- **Census**: In general, census data includes information about population, economy, geography of a nation. Census data records are huge and it needs to be updated frequently as there will be rapid change in data with time. ER techniques are being implemented by Census Bureau of different countries from long period of time. Mostly ER techniques are used in census records to combine the data collected from multiple sources , to match new data with the previously obtained information to form new records of the data [56].

- **Health data**: In order to provide better health care for the patient, hospitals use ER software to improve the quality of the records in patient health information
system. The quality of health data is improved by removing the duplicates in records, restricting the collection of duplicate patient information, integrating previously collected health data of a patient from different records, etc [45].

- **E-Commerce:** ER is referred as product matching in E-Commerce domain. There are several E-Commerce websites that are selling various range of products over the web. Among different websites, the same product in real-world may have different descriptions. In order to compare these products to find out the price variations among them different product matching techniques are applied [36].

- **Others:** Apart from the above specified applications, there are several other applications where ER is applied. Few of them are web information search, national security agencies, bibliographic databases, genome databases, business mailing lists, etc [12, 54]. Also, ER is an indispensable step for data cleansing, data integration, data warehousing and related contents.

### 2.1.3 The Process of Entity Resolution

Figure 2.2 shows a general process of Entity Resolution (ER). There are five main steps in ER. They are data pre-processing, blocking, pair-wise comparison, classification and evaluation [12]. To understand the process of entity resolution, it is necessary to understand each step in detail. The output of the former step is taken as input of the next step as shown in Figure 2.2. The data taken from the data source is pre-processed in the pre-processing step, which is further discussed in Section 2.1.3.1. To avoid comparing each and every record with one another in the pair-wise comparison step, blocking techniques are applied before pair-wise comparison step. Blocking step is discussed in detail in Section 2.1.3.2. To find the duplicates in the data, records in the dataset need to be compared with each other, which is done by the pair-wise comparison step. We discuss pair-wise comparison step in Section 2.1.3.3. The compared records are classified as matches and non-matches using classification techniques, which are discussed in Section 2.1.3.4. The result of the classification step is evaluated using certain strategies in the evaluation step, which are explained in Section 2.1.3.5.

#### 2.1.3.1 Pre-processing

Data pre-processing is an initial step of ER. It is also called data preparation [21], which means preparing the data in a certain manner in order to proceed to the further steps of ER. Often the data sources from where we have to find the duplicates contain unnecessary data for the ER process. If we proceed to other steps without removing this unwanted data, it may result in computational overhead and inaccurate results. Therefore, this step is necessary for improving the data quality and to obtain better results. Methods of Extraction, Transformation and Loading (ETL) are included in data pre-processing step [32]. The pre-processing step can be divided into three stages, namely data parsing, stop words removal and data standardization [21]. They are introduced in detail one by one:
1. **Data parsing:** If a field in the data contains multiple words, they are needed to be located as individual words and processed. This process is called parsing. Example fields where parsing can be applied are ‘Address’ field which contain free-form address, single name field instead of ‘Given name’ and ‘Surname’ fields [30].

2. **Stop words removal:** Frequently occurring words in text, such as ‘and’, ‘is’, ‘the’, etc are called stop words [25]. Stop words do not contain any information that helps us to identify the duplicates from the data. Therefore, these are removed in this step. It is also required to remove special characters like ‘:’, ‘/’, ‘-’, etc. from the data. For example, the ‘Date of Birth’ columns from the tables A and B in Figure 2.1 become easy to compare when the special characters in between them are removed.

3. **Data standardization:** If the data is taken from multiple sources, the schema of the data has to be matched in order to perform ER. Also, the data contained in a specific field should follow a certain format. In data standardization, the schema and data format of the field are standardized. Examples include expansion of abbreviated data, modification of date and time in a specific format, etc [21].

### 2.1.3.2 Blocking

Blocking is also called indexing. The records in the data contain many fields in them and comparisons are performed for almost all the fields in records. Comparison is
2.1. Entity Resolution

computationally expensive step in Entity Resolution. Blocking step aims to reduce the number of comparisons in between the records. Without blocking, comparison step takes $O(n^2)$ comparisons for $n$ records. For example, if there are 100 records for ER process, 10000 comparisons takes place without blocking. Blocking step aims to reduce the runtime of the ER process and accelerate the speed. Records for comparison needed to be reduced by the following two goals:

- Removing the records which are not duplicates.
- Retaining the duplicate records for comparison step.

In the following, commonly used blocking techniques are introduced.

**Standard Blocking:** Standard blocking is a traditional blocking approach in which records with same blocking key values (BKV) are inserted into the same block [12]. After inserting into the same block, the records among the same block are compared to each other. Each record is inserted only into a single block if one BKV is specified. For efficiency, standard blocking is implemented using inverted index data structures. The BKV plays a crucial role in determining the quality and size of the records generated for pair-wise comparison step. Record attributes contain variations in them, multiple blocking keys can be specified to achieve maximum efficiency. The size of the generated block depends on how uniformly BKVs are distributed. Frequency distribution of the BKVs determines the size of the block. Let us assume the distribution of BKV is uniform making the block size uniform. If data from two sources $A$ and $B$ need to be matched and $A$ contains $|A|$ records in it and $|B|$ records in B. ‘$k$’ is the total count of BKVs used. Then the record pairs for $A$ and $B$, $U_{AB}$ generated will be

$$U_{AB} = \frac{|A||B|}{k}$$  \hspace{1cm} (2.1)

For matching within a single data source $A$, $U_A$ generated will be

$$U_A = \frac{|A|}{2} \times \left( \frac{|A|}{2} - 1 \right)$$  \hspace{1cm} (2.2)

Several iterations of blocking step with distinct BKV for each iteration can be done which yields better results for matching, reducing the false matches.

The quality of the records being blocking depends on blocking key and blocking techniques used to block the records. We discuss how to choose blocking key values in the following paragraph.
**Blocking Key Value:** Blocking Key Value (BKV) is chosen by considering the quality, completeness of the data in attribute and the frequency distribution of the attribute \[12\]. Phonetic encoding functions or character-based encoding functions which are discussed in Section 2.1.3.3, are applied to attributes in records to obtain a BKV. The concatenation of phonetic or character-based encoding functions on attributes in the data can be chosen as blocking keys. Multiple blocking keys are used to obtain better blocks of records.

The following are two examples with multiple blocking keys:

- Soundex(SurName)+FirstFiveDigits(PhoneNumber)
- DoubleMetaphone(FirstName)+Soundex(Surname)

**Sorted Neighborhood Blocking:** Sorted Neighborhood is based on the assumption that duplicate records come close to each other in the sorting data step and they are compared in merging step. In this approach, the data from multiple sources is merged before sorting. Records in the data are assigned into different blocks by following three main steps \[29\]:

1. Sorting key: A key is created for each record using the record attributes which is similar to BKV in standard blocking.

2. Sorting data: Records in data are sorted according to the sorting key value. Attributes which are sorted first by the sorting key are given highest preference.

3. Merging: A fixed size window called sliding window is moved through the records, which limits the comparisons for matching records to records in window.

The Sorting key need to be more precise than the BKV in standard blocking as the records which need to be sorted depends upon it. Similar to standard blocking with multiple BKVs, multiple sorting keys can be applied to sorted neighborhood. Several iterations of sorted neighborhood with various sorting keys can be applied for better matching results. This approach is called multi-pass sorted neighborhood blocking. Another approach of sorted neighborhood is to change the fixed window size according to the sorting key used.

**Q-gram based blocking:** Similar to standard blocking technique, the records with same BKV will be compared in q-gram based indexing technique. The difference occurs when the BKV is created. BKV will be created by taking multiple variations of strings using q-grams. q-grams are the sub-strings of length \( q \) of a string. Each blocking key can be converted to a list of q-grams where the maximum length of the \( q \) in the string can be specified by the user. As there are multiple BKVs, the records that are duplicates have higher possibility to be compared with each other yielding better accuracy of results.
2.1. Entity Resolution

The limitation is, as a larger number of records have to be compared with each other, time consumed will be more for large datasets than the previously discussed blocking techniques [13].

Others: The above specified blocking techniques are commonly used blocking techniques for ER process. Apart from them, there are several other blocking techniques. They are suffix array based blocking, canopy clustering, map based blocking, etc [13].

2.1.3.3 Pair-wise Comparison

This step can also be called as record pair comparison step. In ER, the fields or strings need to be compared and then the similarity between them is measured. The techniques used for comparing strings are called similarity measures or field matching techniques. Usually, differences in the strings occur due to typographical variations in them, which are caused by numerous reasons, such as misplaced characters, missing characters, missing fields, typographical errors, etc. Similarity measures are divided into two types, namely sound-based similarity measures and character based similarity measures [21]. In pair-wise comparison step, sound-based or character-based similarity measures can be applied to calculate the similarity between records. In most of the cases, character-based similarity measures are applied in pair-wise comparison step and sound-based similarity measures are applied for blocking techniques in ER. In this section, we discuss various types of sound and character-based similarity metrics [21].

Sound-based similarity metrics: Sound-based similarity measures are used for calculating phonetic equality between two strings. If two strings sound similar, then they are said to be phonetically equal strings. These techniques are applied mostly on the real world names. All the sound based similarity techniques attempt to convert the word to a code based on the way it is pronounced [11]. There are various techniques to find the phonetic similarity of strings in which three of the commonly used techniques are discussed briefly in this section.

1. **Soundex:** Soundex is the commonly used algorithm to measure phonetic equality among strings. It is designed to be applied on the English names. To convert a name to its equivalent Soundex code, several rules are to be followed. Soundex rules [37]:

   - Preserve the first letter of the name as prefix of the code.
   - Ignore the occurrences of a, e, i, o, u, w and h in all other positions.
   - Convert the remaining letters by following the rules in Table 2.1.
   - If there are similar occurrences of the same number after conversion from table, consider only the first occurrence of the code.
Table 2.1: Rules of soundex string conversion [11]

<table>
<thead>
<tr>
<th>Text</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>b, f, p, v</td>
<td>1</td>
</tr>
<tr>
<td>c, g, j, k, q, s, x, z</td>
<td>2</td>
</tr>
<tr>
<td>d, t</td>
<td>3</td>
</tr>
<tr>
<td>l</td>
<td>4</td>
</tr>
<tr>
<td>m, n</td>
<td>5</td>
</tr>
<tr>
<td>r</td>
<td>6</td>
</tr>
</tbody>
</table>

- Limit the resulting code with one alphabet (prefix) and three numbers. If the converted result is less than the limit, then add 0 to match the code limit and if code contains more characters, remove the extra numbers.
- Example: Soundex of ‘Sravani’ is ‘S615’.

2. **Metaphone**: Metaphone algorithm is suggested as a better encoding algorithm for English names than Soundex [21]. In this algorithm, the alphabets are replaced by the code with other alphabets. Several rules for the conversion into Metaphone include:

- Retain the first letter and ignore the vowels after the first letter.
- 16 constants replace the English alphabets except the first letter [37].
- Example : Metaphone of ‘Sravani’ is ‘SRFN’.

3. **Double Metaphone**: Double Metaphone algorithm [46] is an improved version of Metaphone algorithm. Previously discussed sound-based similarity measures are developed for English names, whereas Double Metaphone algorithm can be applied for Asian and European names [11]. The result of the double metaphone algorithm will be in the form of two encoded strings, which makes results in better comparison of strings.

- Broad number of rules are applied on the name string to convert it into two encoded strings [46].
- Example : Double Metaphone of ‘Sravani’ yields ‘SRFN’ and ‘SRFN’, ‘Culzewski’ yields ‘KLSS’ and ‘KLTS’.

**Character-based similarity metrics**: To compare the records or fields after the blocking step to find the similarity between them, character-based similarity techniques are applied [21].

1. **Levenshtein edit distance**: There are various edit distance based algorithms and one of the oldest among them is Levenshtein edit distance [44]. Levenshtein edit distance is calculated in between the two strings $A$ and $B$ as the number of
2.1. Entity Resolution

The edit-operations for this technique include:

- Insertion of a character into the string.
- Deletion of a character from the string.
- Substitution of a character with another character in the string.

The cost of each edit operation is one. Higher the number of edit operations, lesser the similarity between the strings. The dynamic programming algorithm [44] is used to compute the edit distance for two strings. It takes $O(|A||B|)$ time with the strings of length $|A|$ and $|B|$ [21]. The distance $d$ which is calculated ranges in between $0 <= d(A,B) <= max(|A||B|)$.

Example: If we apply Levenshtein edit distance to two strings ‘calculates’ and ‘calculator’ the result will be 2, i.e. it takes 2 edit operations to transform one string to another. The result will be ‘0’ if we compare strings with same characters.

2. **Hamming distance**: In Hamming distance method, only one edit operation i.e. substitution of the character with another character in the string is allowed. The distance is calculated by adding all the substitutions required to convert a string $A$ to string $B$. The calculated distance ‘$d$’ ranges in between $0 <= d(A,B) <= |A|$ [44].

Example: Hamming distance between strings ‘data’ and ‘dates’ will be 2 as we need to convert ‘a’ in $A$ to ‘es’ in $B$, which takes 2 substitutions. If we compare hamming distance between ‘1000’ and ‘1111’, result will be 3.

3. **Jaro and Winkler distance metric**:  

**Jaro distance metric**: It was introduced by Jaro, to compare the similarity between names. Let us consider two strings $A$ and $B$, with $|A|$ and $|B|$ as lengths of the strings. Let ‘$c$’ be the common characters for $A$ and $B$ within certain window positions. ‘$t$’ be the half of transformations to convert a string $A$ to string $B$. The jaro distance between two strings is measured as [59]:

$$Jaro(A, B) = \frac{1}{3} \left( \frac{c}{|A|} + \frac{c}{|B|} + \frac{(c - t)}{c} \right)$$

(2.3)

Example: If we compare Jaro distance between ‘book’ and ‘bunk’, the similarity score will be 0.67 if calculated according to Equation 2.3.

**Jaro and winkler distance metric**: According to the previous studies [48], most of the errors occur in the middle and end of the strings than that of the beginning. In Jaro winkler distance metric, a few changes are made to the Jaro distance metric.

Common modifications include:

- Higher preference to the prefix: If the prefix of two strings is same, similarity between them is increased.
• Winkler algorithm is applied to the basic jaro distance metric to increase the similarity in between the strings [59].

\[
SimWinkler(A, B) = Jaro(A, B) + \frac{p}{10} (1.0 - Jaro(A, B))
\]  

(2.4)

where \( p \) is the length of prefix ranging from 1 to 4.

Example: JaroWinker distance metric for strings ‘book’ and ‘bunk’ is 0.70, where value of \( p \) is 1. A few other modifications are made on the jaro distance metric which are based on the similar characters, and handling longer strings [59].

4. **Exact Similarity Metric:** In this metric, the strings match only if all the characters in it are equal. A and B strings match and the result will be 1 only if \( A=B \) or else, if \( A\neq B \) the result will be 0 [12].

Example: For two strings ‘apple’ and ‘apply’ the exact similarity will be 0. For ‘2220’ and ‘2220’, the exact similarity metric will be 1.

It is to be noted that the values of the pair-wise comparison will be based on the algorithm that we use to compare the strings. The result value will vary with various algorithms. The comparable fields must be analyzed and based on the type of the strings, proper algorithm should be chosen.

2.1.3.4 **Classification**

The compared records in the pair-wise comparison step are classified as matches and non-matches by classification techniques. This is called binary classification problem. We will discuss about threshold-based, rule-based, learning-based classification techniques to classify the compared records.

**Threshold-based Classification:** In this classification technique the sum of similarities between all the attributes in pair-wise comparison is calculated as \( TotalSim \) [12]. A similarity threshold is applied on \( TotalSim \) so that the class of record-pair (match or non-match) is decided according to the threshold given. Let \((r_i, r_j)\) be a record pair which are compared and \( TotalSim \) is calculated for it. Let ‘\( t \)’ be the similarity threshold applied to record-pair \((r_i, r_j)\). The record-pair is classified as below.

\[
TotalSim(r_i, r_j) \geq t \rightarrow (r_i, r_j) \text{ is a match}
\]  

(2.5)

\[
TotalSim(r_i, r_j) < t \rightarrow (r_i, r_j) \text{ is a non-match}
\]  

(2.6)

In threshold-based classification, all the attributes are considered in the same way, without giving importance to the distinct and special attributes. To overcome this limitation, a weighted threshold can be calculated by giving specific weight to each attribute depending on their importance [12]. The detail information of the individual similarities of record pairs is lost in this technique. Other classification techniques can also be applied to overcome this issue in threshold-based classification.
2.1. Entity Resolution

**Rule-based Classification:** In rule-based classified technique, certain rules need to be specified. The records will be classified into matches and non-matches based on the given rules. Rules are based on the similarity values obtained by the record fields and they are combined using conjunctions, disjunctions, and negations. Consider a record pair \((r_i, r_j)\) and \(P\) is the predicate on record pair, \(C\) is the result of classification. Then, rules applied will be in the form of \(P \Rightarrow C\). It is possible to apply rules manually or learn from the training data [12].

**Learning-based Classification:** Learning-based classification techniques from the areas of data mining and machine learning are applied for classifying the records for ER [27]. In learning-based techniques, the algorithm learn from the training data given to it. The training data contains true match and non-match labels for the records. In machine learning terminology, the similarity values obtained from field comparison techniques are called feature vectors or parameters. The feature vectors from the train data are selected so that the classifier accurately classifies the record pairs as matches and non matches. Before classifying the new record pairs, the accuracy of the classifier is tested and tuned to obtain high accuracy. We will discuss commonly used learning-based classification techniques.

1. **Decision Tree:** Decision Tree Classification is one of the most commonly used algorithm for supervised learning classification [27]. It works in a top-down approach. Decision tree contains three kinds of nodes. Namely, root node, decision node and leaf node. The training data is taken at the root node. The attributes in the training data are divided and splitted by making certain decisions and rules in the decision nodes. Tests are performed on attributes in the decision node. Every node except the root node in the tree is the result of the previous node. The decision nodes will continue to divide until there is no further decision to make and all the attributes are splitted. They stop dividing if all the attributes in it belong to the same class. If there are no further steps to perform for the decision node, leaf node is assigned to it with the class label (match or non-match) [27]. Example of the Decision tree is illustrated in Figure 2.3. In the figure, \(\text{Sim}\) is the field similarity measure applied on the attributes.

2. **Random Forest:** The accuracy of the classifier to predict the classes increases if the combination of classifiers increase. Random forest is a collection of decision tree classifiers [9]. In this algorithm, random subsets are created from selecting random attributes from the training data. Using these random subsets, multiple decision trees are created, which are called random forests. As there are many decision trees to choose predicted labels in the random forest, voting is conducted for each decision tree and the class of the record pair is predicted based on the results of voting. Example of Random Forest algorithm is illustrated in Figure 2.4.
2. Background

Figure 2.3: Example of decision tree classification

Figure 2.4: Example of random forest classification
2.1. Entity Resolution

<table>
<thead>
<tr>
<th>Classification</th>
<th>Match</th>
<th>Non-match</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gold standard</td>
<td>TP</td>
<td>FN</td>
</tr>
<tr>
<td>Non-match</td>
<td>FP</td>
<td>TN</td>
</tr>
</tbody>
</table>

Table 2.2: Confusion matrix for evaluating the classified records

2.1.3.5 Evaluation

Evaluation is the last step of ER. The correctness of the classified records as matches and non-matches is calculated in this step. Evaluation measures are necessary to calculate the quality of obtained results. For all the evaluation measures, the gold standard data of the records is needed [12]. Gold standard data is the data which contains the information of the correct matches and non-matches of the records. Using this data we can know the accuracy of the classified records.

Based on the classification of records we the can assign them to four categories [15]:

- True Positives (TP) - The total number of records, which are classified as matches and they are true matches when compared with the gold standard data.
- False Negatives (FN)- Total number of records, which are classified as non-matches but are true matches which belong to the same entity.
- True Negatives (TN)- True negatives are the total number of records, which are classified as non-matches and are true non-matches.
- False Positives (FP)- Total number of records, which are classified as matches but they are non-matches when compared to the gold standard data.

All these four categories are represented in the form of a matrix named confusion matrix. Quality measures like recall, precision and F-measure are calculated to evaluate the classification of records, based on the categories illustrated in Table 2.2. These metrics are widely used for information retrieval [41].

**Precision:** It is also called positive predictive value. Precision is the ratio of true matches (TP) among the classified matches (TP+FP). It calculates the correctness of the classifier in classifying records as true matches [15].

\[
Precision = \frac{TP}{TP + FP}
\]  
(2.7)
Recall: Recall is also known as true positive rate or hit rate. Recall is the ratio of actual matches (TP+FN) in gold standard, which are classified correctly as TP by the classifier. It is calculated in order to know, the number of record pairs that are actual true matches are being classified as true matches by the classifier [15].

\[
\text{Recall} = \frac{TP}{TP + FN}
\] (2.8)

F-measure: It is also known as F-score. F-measure is the harmonic mean of recall and precision. The value of F-measure increases with the increase in recall and precision values. To obtain better F-measure, precision and recall need to be balanced and high [15].

\[
F - \text{measure} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}
\] (2.9)

2.2 Parallel Entity Resolution

In this section, we discuss the basic concepts of parallel ER. ER with the large scale data is an expensive process, which consumes large amounts of time [35]. It also requires resources with higher capabilities of CPU and RAM. Although the efficiency of ER can be improved by blocking step, the number of record pairs need to be compared are so large that the execution time may be still unacceptable when the input volume is quite large. To decrease the costs of ER process and increase efficiency, it can be performed in a parallel, distributed or cloud environments. Using parallel techniques, the performance of ER process will be improved as the tasks are divided and processed in parallel [33]. In Section 2.2.1, basic knowledge of parallel computing is discussed, which includes the definition, motivation and the performance metrics of parallel computing. In Section 2.2.2, we discuss why ER is suitable to be performed in parallel. Spark is applied to implement our parallel ER in this thesis. Therefore, in Section 2.2.3, we will introduce the knowledge of Spark.

2.2.1 Parallel Computing

Motivation and Definition: In traditional computing (serial computing), a program is written using a series of instructions which are executed one after the other. The program can only be executed on a single processor. The instructions have to wait for the others to execute as only one instruction have to be executed at once. Depending upon the complexity and instructions given in program, it sometimes takes up to several minutes, hours or days to run the serial program. In parallel programming, multiple resources such as executors or cores are used to overcome the complexity caused when using a single processor for serial program. The instructions or tasks will be divided among the resources and they are executed in parallel. Generally, parallel computing can be implemented by the use of various resources. It can be implemented in a single machine which has multiple cores or processors in it. Another way of implementing
parallel computing is to use multiple machines such as clusters, grids or executors. It is also possible to use parallel computing frameworks which are designed to support parallelism [5].

**Performance metrics for parallel computing:** To calculate the parallel performance of parallelism of the application, following performance measures will be used [55]:

- **Speed-up:** Speed-up ‘$S_n$’ measures the ratio of the total time taken to run the application on one node and the total execution time to run parallelly in the cluster using ‘$n$’ nodes.

  \[ S_n = \frac{T_1}{T_n} \]  

  $T_1$ : Time taken to run application on one node  
  $T_n$ : Time taken to run application using ‘$n$’ nodes

- **Efficiency:** Efficiency ‘$E_n$’ is dependent on Speed-Up. It is the measure of usage of utilization of processors in a parallel application.

  \[ E_n = \frac{S_n}{n} \]  

- **Scale-out:** It is used to indicate the ability to handle increasing amounts of data with increasing the cluster size by increasing the workers in parallel environment. Ideally, the execution time should be constant with increasing amounts of data and workers.

### 2.2.2 Parallel ER in a Cluster

In this section, we give a short overview on parallel ER in a cluster. In ER, most of the steps are independent to each other and they can easily be separated and performed in parallel. Parallel ER is mainly data parallelism, which divides the input data into several partitions, then each partition is assigned to one node. The nodes communicate with each other and sometimes the result of some steps will get shuffled from one node to another node, if the result of one node is needed to the another node.

### 2.2.3 Apache Spark

In our parallel ER application, we use Apache Spark to help us with its implementation. Therefore, we provide in this section all required foundations to understand Apache Spark in our application.
Apache Spark and Usages: Apache Spark is an open-source engine for big data processing using cluster computing. The first paper on Spark is written by Matei Zaharia in 2010. Spark API was initially released in May 2014. From then, the development of the project was very active and there were many stable releases of Spark with Apache License 2.0. Berkeley’s AMPLab [31] in University of California, Apache Software Foundation and DataBricks community are the active developers of Apache Spark. According to the official website of Spark [23], “Apache Spark is a fast and general-purpose cluster computing system”. It provides an interface to run the applications in parallel and in a fault-tolerant way. Spark is well known for its speed and ease of use. Using spark, the computations can be performed in-memory. It is possible to run programs interactively in a shell and it is also possible to build stand-alone applications. This increases the speed and efficiency of applications. Spark provides APIs for Java, Scala, Python, and R programming languages, which makes it easy for developers to use Spark. The runtime of similar applications using Spark and Hadoop are compared in Figure 2.5. Another benefit of using Spark is the code to develop an application in Spark is less than the other frameworks, as illustrated in Figure 2.6.

Figure 2.5: Comparison of runtime of logistic regression application between Hadoop and Spark [31]

Figure 2.6: Code size of Spark compared with other frameworks [23]
Spark Core: Spark core is the engine of Spark which schedules, distributes and monitors jobs (tasks) across a cluster. Other components/libraries can be run on the top of the Spark core, which can be used combinedly or independent to each other. If Spark core is modified and optimized, the benefits are passed to the top level components (libraries) of Spark. The spark core and its components together called Spark unified stack, are shown in Figure 2.7. It is possible to use an interactive shell in Spark, where the data can be analyzed interactively [23]. In the following, we introduce the important concept of Resilient Distributed Dataset (RDD), which is an immutable primary abstraction of data in Spark core.

RDD: Resilient Distributed Dataset (RDD) is immutable primary abstraction of data in Spark core [61]. RDDs are data elements which are operated parallelly over nodes in a cluster. In Spark, a lineage tree called Direct Acyclic Graph (DAG) is constructed by keeping track of the RDD operations, which is evaluated by running an application. DAG is updated each time when a transformation is called. If a node fails in a cluster, it can start over by reconstructing from where it leaves using DAG. In Spark, RDDs can be created in two ways [23, 61]:

1. By Parallelizing an existing collection from driver program or referencing a dataset in an external storage system, such as a shared filesystem, HDFS, HBase, etc.
2. By transforming, caching or persisting an existing RDD to another RDD.

More operations can be performed on RDDs than MapReduce. RDD operations can be divided into two types, namely transformations and actions[60].

Transformations: They create a new RDD from an existing one. In general, transformations are lazily executed, i.e. RDD is only transformed when it encounters an action on it.

Examples of most commonly used transformations include:

- filter(): Applying filter operation on a function f, returns a new dataset by selecting the elements specified in f.
- map(): It returns a new dataset by passing all the elements in the source through the specified function f in the map operation.
- distinct(): It returns distinct elements as a new dataset from the previous dataset.
- union(): It takes dataset and the argument as input and returns a new dataset with the union of the elements.

Actions: Actions return a value to the driver program by executing the transformations. Examples of most commonly used actions include:
count(), reduce(), collect(), first(), saveAsTextFile(path), etc.

**count()**: It returns the total number of elements contained in the dataset.

**reduce()**: It combines the elements in the dataset, using a specified function $f$.

**collect()**: It returns the elements of dataset in the form of an array to the driver program.

**first()**: It returns the first element in the dataset.

**saveAsTextFile()**: It saves the dataset as the textfile if the path is specified.

**RDD Persistence**: RDDs can be persisted so that they can be stored in memory and it is possible to reuse them in other operations of the application. This feature increases the speed of the future operations. Different storage level options such as cache, persist, disk can be applied on RDDs. If the application does not fit in memory, persisted RDDs can be stored on disk for computations.

**Spark components/libraries**: Spark has four main components which are helpful to process scalable data in parallel. The top level components are [23]:

![Spark unified stack](image.png)

Figure 2.7: Spark unified stack [23]

- **Spark SQL**: Spark SQL is developed for the relational data processing in Spark. Spark SQL operations can be performed on the data with known schema (structural data) or the data where the schema can be specified explicitly (semi-structured data). SQL Context is loaded to start the Spark SQL, where the data can be handled as tables containing rows and columns. Regular SQL and Hive QL queries can be performed in parallel on the huge amounts of data. Spark SQL provides an ability to intermix the relational and procedural code in the applications making its use sufficient for various big data applications. Spark SQL benefits from its two main features. They are [4]:

  **Dataframe API**: Spark SQL provides the programming abstraction called Dataframe API. It was named as SchemaRDD in the early development as it is similar to Spark RDD, but with a schema. Dataframe API is similar to Python pandas and R, which are mostly used for data analytic jobs. The data is stored in columns giving high impact than the java or python object stored in RDD. All
relational operations and high-level operations like join, filter, group, sort that takes higher computation time can be performed in an efficient manner using DataFrames than RDD. DataFrames are more expressive, convenient and efficient than RDD API in most of the situations. User Defined Functions (UDFs) are also supported, where it is possible to define our own functions other than built-in functions, which is necessary to built any specific application. As illustrated in Figure 2.8, DataFrame improves the performance of the application, irrespective of the programming language used, compared to RDD.

Figure 2.8: Comparing runtime of aggregation query using RDD and DataFrame API with various programming languages [3]

DataFrame Operations: Various operations can be performed on a DataFrame by using DataFrame API. Some examples of DataFrame operations which we used in our thesis include [23]:

*Data Frame Operations:*
count(), cache(), drop(), repartition(), withColumn(), etc.

**count()**: Applying count operation on a DataFrame \( d \) returns the total number of rows in \( d \).

**cache()**: Applying cache operation on a DataFrame \( d \) caches \( d \) in-memory.

**drop()**: Drop operation of a column \( c \) on a DataFrame returns a new DataFrame without column \( c \).

**repartition()**: Repartition operation with an integer \( i \) returns the DataFrame with \( i \) partitions.

**withColumn()**: withColumn operation of a column \( c \) on a DataFrame returns a new DataFrame by adding a new column \( c1 \) or by replacing the existing column \( c \).

**Catalyst Optimizer**: The syntax of DataFrame is captured as tree shaped logical plan, they are converted to physical plan and then the results are computed by catalyst optimizer. Catalyst Optimizer in Spark SQL provides optimized execution of queries before its execution, with the use of the catalyst framework the applications are able to run faster. It supports rule-based and cost-based optimization. Catalyst works internally in Spark to support various data sources and advanced analytics in Spark SQL. It uses the features of Scala programming language to define various rules for code optimization.

Using Spark SQL, it is possible to load data from various external sources such as JSON, Cassandra, MySQL, Hadoop etc. Advanced analytic jobs like streaming, machine learning, graph processing tasks are supported using DataFrames. DataFrame API work on top of all the Spark components making them easy to use by giving similar performance for Java, Scala, Python and R as shown in Figure 2.9. As Spark SQL provide all these benefits, DataFrame API has been chosen for our Spark-based ER application instead of RDD API.

- **Spark MLlib**: Supports various iterative for algorithms machine learning to be performed in the distributed environment.

Spark supports various machine learning algorithms to be performed in the distributed environment. Machine Learning (ML) pipelines are built on the top of Spark SQL, using its DataFrame API, whereas MLlib uses RDD API. Currently, according to the Spark official website [23], DataFrame-based API is announced as the primary API for machine learning due to its ease of usage and benefits of Spark SQL. It is possible to access all the algorithms for an application by switching from ML to MLlib or vice versa by converting DataFrame to RDD. Main techniques of Spark machine learning include Classification, Regression, Collaborative Filtering, Clustering, Dimensionality Reduction, Extraction and Transformation of features [42].

ML API is based on ML pipelines, which consists the work flow of data in them. In ML pipeline, transformation of data takes place in each step. Each step takes the input data set, applies transformation algorithm required for the application
and outputs the transformed data set. The transformed data set is taken as the input for the next stage in ML pipeline. As ML pipeline is based on Spark SQL API, it has the benefits of optimized execution, various operations on columns and user defined functions. The ML transformations include feature extraction, normalization, training a model using required ML algorithm and validating the classifier. Parameter tuning is supported in ML pipelines for increased accuracy of the machine learning algorithm. The basic workflow of ML pipeline is shown in Figure 2.10. When the training data is loaded, the features are transformed using various built in API for feature extractors, transformers and selectors. Feature extraction contains API for extracting features from unprocessed data. Feature transformation has various algorithms for modifying and scaling features. Feature selection means selecting a small subset of features from data with many features. It is possible to use multiple transformers in a single ML pipeline depending upon the application. After feature extraction, the features are given to train the model using a machine learning algorithm. Once the model is trained, it can be evaluated for accuracy and new data is applied on the model [58]. With all the above considerations, in parallel ER application, we chose Spark ML pipelines for the classification step.

- **Spark Streaming:** It is used for real-time data analysis and processing tasks.

- **GraphX:** Used for distributed graph computing. It uses Resilient Distributed Graph (RDG) as primary abstraction. Data is represented in the form of vertices and edges. It supports only Scala language API as of now.
As we did not use Spark Streaming and GraphX in our application, we only introduced them briefly. In the following, the procedure to execute an application in a Spark cluster is introduced.

![Cluster execution framework in Spark](image)

**Figure 2.11: Cluster execution framework in Spark [23]**

**Executing an application in a Spark cluster:** A Spark application can run in local and cluster mode. Generally, local mode is used for testing and implementation of the application and cluster mode is used for large-scale data processing of the application. The spark cluster framework works as shown in Figure 2.11. In cluster mode, Spark uses master/slave architecture. It is possible to run Spark application as stand-alone Spark application, which uses its own built-in scheduler for scheduling jobs. Other than standalone Spark cluster, the Spark application can run on Hadoop clusters such as Apache Mesos and Hadoop YARN. There exists driver program in the master node which coordinates the workers called executors. The cluster manager is responsible for allocating the resources across application in the cluster. A Spark application runs combinely on master and executors. The cluster manager in Spark launches the application on the workers which are in distributed environment. In the Spark driver, the program is divided into smaller units for execution called tasks. The tasks are divided between the executors and the executors run their given tasks individually. Once they finish the given task, they return the results to the driver. The executors stay alive until the end of the application. Even if an executor fails in between, the
application continues running in the cluster. The running Spark application can be monitored using a web User Interface (UI). In web UI, the jobs of the executors, stages of the jobs (tasks), cluster environment, active and inactive executors, the metrics of an executor and the application logs can be checked. Generally, any Spark application will be initiated by calling SparkContext in the program. SQLContext is loaded after calling the SparkContext. After initiating the Spark SQL context, the data can be loaded.
2.3 Summary

In this chapter, we provided the background information needed to understand the process of ER and parallel ER. In the first section, we described the concepts of ER and the steps taken to carry out the process of ER. Common techniques involved in pre-processing, blocking, pair-wise comparison, classification and evaluation for ER are explained briefly. In the second section, we introduced the concept of parallel entity resolution and the measures taken to calculate the parallel application. We discussed about the ER in parallel environments. Finally, features of Spark, components of Spark and the way an application works in the cluster environment in Spark are discussed.
3. Related Work

Related work regarding parallel Entity Resolution (ER) is discussed in this chapter. In Section 3.1, parallel ER without using any programming model will be discussed which is called general parallelism. MapReduce-based ER techniques will be discussed in Section 3.2. Spark-based parallel ER techniques will be discussed in Section 3.3.

3.1 General Parallelism

Several approaches are proposed in order to reduce the processing time of ER using distributed environment [6, 14, 17, 34, 40, 52], by increasing the number of cores or processors or memory. These approaches focus on improving the efficiency and performance to detect duplicates in the ER applications. D-Swoosh [6] contains group of algorithms which are developed for performing distributed ER using multiple processors. Febrl [14], is a parallel open source system for entity resolution. In [17], an approach for parallel ER is implemented using multicore processors. [40] is based on ER using graph-based parallelism. There are other parallel ER approaches, namely dedoop [34] and parallel set-similarity joins [52], which are based on MapReduce programming model in Hadoop, we discuss them in Section 3.2.

3.2 MapReduce-based Parallelism

Dedoop is a tool developed by Lars Kolb [34] for Entity Resolution on distributed environments. It uses MapReduce framework for execution of entity resolution tasks in parallel. It provides the web user interface, which allows a user to choose various approaches of ER. Among variety of similarity measures, blocking techniques, and classification measures required techniques can be chosen and specified ER tasks are executed on Hadoop cluster accordingly. A single or multiple files from where the duplicates can be matched must be given as input to Dedoop from where the records are matched. Dedoop provides different types of phonetic and field-based similarity metrics
to choose for blocking and comparison steps. Blocking techniques can be chosen between standard blocking and sorted neighborhood to reduce the search space in comparison. Classification techniques can be chosen from rule-based, threshold-based and machine learning-based classification methods. The problems of running distributed tasks are addressed in Dedoop by providing advanced techniques such as redundancy-free matching and load balancing to run ER in parallel. Redundancy-free matching is obtained by a multi-pass blocking technique where multiple blocking keys are used which helps in increasing quality of records after blocking. In their work, load balancing techniques are used to avoid data skew, so that the data can be allocated evenly among the virtual machines in the Hadoop cluster. As the project is not open-sourced, the execution workflow of map tasks and reduce tasks are not known when ER process is executing in the cluster.

Set-Similarity join is similar to ER, which is used to find the duplicate records. It is the technique where various records or text from various files are joined when they meet a certain similarity threshold. The similarity threshold is calculated using various string similarity metric operations. Distinct techniques are proposed to perform set-similarity joins efficiently. Gravano [26] proposed techniques for joining the text for RDBMS data using SQL queries. Various string metrics are used using SQL and evaluated in his work. Chaudhuri [10] used overlap operators in SQL for set-similarity joins, functional dependencies which are applied to the subset of data. They implemented filtered SS join where records are joined when the prefix of subsets are matched. Parallel set-similarity joins are implemented effectively using MapReduce framework which in [52] and [18]. Various distributed environment issues like data partitioning, self-join, filtering are mainly researched while implementing in MapReduce framework.

### 3.3 Spark-based Parallelism

As the name of the title “A Spark-based workflow for probabilistic record linkage of healthcare data” suggests, the work in [47] is based on the Spark framework to implement entity resolution. Their work is focused on the large scale databases containing healthcare data. Spark-based system for record linkage (ER) was mainly used in order to obtain accurate data marts. Their work flow is divided into four stages. In the first stage, the databases are analyzed using SPSS software, which is a statistical tool. The next stage is preprocessing, where they used Spark as it is a data intensive task. In order to pair the similar records, deterministic and probabilistic algorithms are performed where data marts are produced. In the last step, to evaluate the accuracy of the data marts statistical tools like SPSS and stata were used. Also in their work, to support the confidentiality of the data, bloom filters were used. In order to evaluate their approach, an OpenMP interface [1] implemented and compared with the Spark-based approach.

In [53], entity resolution is implemented using Spark using Adverse Drug Reaction (ADR) databases. ADR reports are databases maintained by the drug regulators and World Health Organization (WHO) in several countries [53]. In their method which uses Spark, k-Nearest Neighbor (kNN) classifier is used to classify the records. For
evaluation of their approach, Support Vector Machine (SVM) algorithm is used, in order to compare the results with the KNN based approach.
3. Related Work
4. Spark-Based Entity Resolution: Concepts and Implementation

We discuss concepts and implementation of our parallel Entity Resolution (ER) application using Spark in this chapter. In Section 4.1, we discuss the motivation of using Apache Spark to implement the parallel ER application. In Section 4.2, we discuss the implementation choices made to implement the application. The workflow and implementation details of our application are discussed in Section 4.3. In order to increase the performance of application, we applied tuning techniques, which are discussed in Section 4.4.

4.1 Motivation

As mentioned in the last chapter, parallel entity resolution is required to improve the efficiency. In recent years, Apache Spark is one of the most active frameworks used for data intensive tasks, which provides more programming operations and features than Hadoop. It is also possible to combine Hadoop with Spark. We used Apache Spark as the distributed computing framework. Its features such as in-memory computations, in-built libraries are used for efficient implementation of ER application. Currently, most of the research work is focused in the use of MapReduce programming model. However, MapReduce has its own drawbacks compared to Spark. Next, MapReduce and its drawbacks are introduced.

MapReduce and its Drawbacks

MapReduce: In order to process the large amounts of data, Google introduced an algorithm which follows the divide and conquer rule named as MapReduce [20]. MapReduce programming model lets the users specify two functions, namely map and reduce. Map function contains a key/value pair which generates the intermediate set
of key/value pairs. The reduce function combines all the intermediate values which contain the same key \([20]\). It provides high tolerance and scalability for the applications. Apache Hadoop is an open-source framework for data processing which is developed based on Google’s MapReduce algorithms. Java applications can be built using map and reduce tasks in Apache Hadoop. Hadoop framework includes four modules \([22]\):

- Hadoop Common: These contain the basic utilities which are used for the other three modules. They contain libraries for I/O abstraction and configuration scripts to start Hadoop.

- Hadoop YARN: YARN refers to Yet Another Resource Negotiator, which can be used for job scheduling and managing resources in a cluster. YARN manages the resources by assigning tasks to resource manager and node manager. YARN can also be used by other cluster computing frameworks other than Hadoop.

- Hadoop MapReduce: MapReduce-based programming model used for parallel processing of large datasets.

- Hadoop Distributed File System (HDFS): HDFS provides distributed storage to large datasets in the cluster. It makes data highly available to the cluster, supports minimal data notion (shuffle) and rack awareness.

**Drawbacks:**

Though Apache Hadoop is highly successful Big Data technology, its operations, map tasks and reduce tasks carries disk operations (read and write) for every task, which needs more disk space for processing their jobs. Mostly, these systems are built around acyclic data flow model, which are not suitable for some applications \([61]\). When its batch jobs are combined with other systems to fit the use case of an application, additional computational overhead is added to it. Applications can only be programmed using Java, making it hard for the other language developers to learn and use. These limitations can be overcome by using Apache Spark framework.

As mentioned above, compared with MapReduce, Apache Spark does not have such drawbacks of MapReduce: Apache Spark supports cyclic data flow and both in-memory and disk computing, which make it possible to run programs 10-100 times faster than MapReduce. People can write applications quickly in Java, Scala, Python, R and use it interactively from the Scala, Python and R shells. In addition, it powers also a stack of libraries: Spark SQL, Machine learning library MLib, Spark streaming, GraphX to make it possible to combine these libraries seamlessly in the same application. Due to all above mentioned factors, Spark is chosen as the tool for parallel ER.

### 4.2 Implementation Choices

In this section, we discuss briefly about the programming language used, the development environment in which we developed our application, dependency plugin used, and the information about the external libraries that we used in several stages of our application.
4.2.1 Programming Language

We used Java programming language with JDK-8, which supports the use of lambda expressions. The performance of scala and java are the same as they run on Java Virtual Machine (JVM). In RDD API, the performance of application can be affected by the programming language used. Python takes much runtime, as it has a different execution environment. However, in DataFrame API the implemented programming language does not affect the performance of the query. The only limitation of using java is it does not support interactive analysis of the application, which is supported in scala and python by Spark shell.

4.2.2 Development Environments

Eclipse IDE with maven dependency plugin is used for developing the application. All the libraries and dependencies used for the application must be specified in Project Object Model (POM) file in maven. To run our application in the cluster, the code along with the dependencies are shipped into the jar file.

4.2.3 External Libraries

External libraries for string comparisons are used in our application, as Spark is inbuilt with only few string comparison techniques like soundex and cosine similarity. Rockymadden stringmetric library [39] is used for comparing strings in blocking and field comparison steps. We used Apache Language Codec library [24] for Double MetaPhone, as rockymadden stringmetric do not support Double MetaPhone technique. For parsing Comma Separated Value (CSV) file, we used Data Source API from databricks [19], named spark-csv which is an official Spark API to read CSV files.

4.3 Workflow of Spark-Based Entity Resolution

Our parallel ER application workflow is illustrated in Figure 4.1. As we have already discussed in Chapter 2, With our application, it is possible to decide whether to execute the preprocessing step or not before using Spark framework. The next step is loading the data into Spark SQL as DataFrame. The blocking is performed on the loaded DataFrames in the next step. Field comparisons are performed for the resulting DataFrames in blocking. In the classification phase, it is possible to choose from three classification algorithms: decision tree, random forest and threshold-based classification. The result of classification stage will be saved as DataFrames, which shows the match and non-match status of records. The classified records are evaluated as true matches, false matches, true negatives and false negatives in the evaluation stage. All the steps in our application can be performed individually by saving the result of each step and taking the output from the previous step. It is also possible to run all the stages together in the application, without taking disk inputs and outputs into intermediate stages. The second method obtains high performance from Spark as all the codes gets optimized before executing in Spark SQL component. The implementation details of ER application using Spark components are discussed in the following sections.
4.3.1 Pre-processing

Pre-processing is the initial step for entity resolution process. It is an optional phase for our application. The use of this step depends upon the data from which we want to find duplicates. If the data is dirty, i.e. if it contains frequent stop words, special characters, blank lines, it is necessary to remove them before advancing to the next stage. The
columns with long sentences are parsed individually and trimmed. The output of this phase will be in the same format as the input source. The preprocessed data is saved as an input for the following steps.

4.3.2 Loading data

In general, it is possible to load data with various formats and sources in Spark SQL. As we work with data in CSV format, databricks CSV library is used for loading the data. The Schema of the data is specified manually and the data is loaded with its header into DataFrames. The data can be analyzed after loading into DataFrame, by the use of first(), show() or describe() methods in DataFrame API [23]. The first() method, shows the first row of DataFrame. First 5 rows are displayed in console using show() method and describe() displays the statistics of the specified columns in DataFrame. In this step, further pre-processing can be done after loading as DataFrame and analyzing it. For our application, we eliminated two columns that have several null values in them, as they carry additional overhead in processing. The null values in DataFrame, which are empty fields, will be handled by inserting zeros into them.

4.3.3 Blocking

Recall that, as described in previous chapters, blocking is performed to reduce the overhead of comparing every record with each other. We followed the approach of standard blocking with the use of Spark SQL. In order to specify the blocking key, we used phonetic similarity measures to transform the original string to phonetic code. Phonetic similarity measures are defined using User Defined Functions (UDFs). In our application, it is possible to choose various phonetic similarity algorithms such as Soundex, which is already inbuilt in Spark SQL, Other phonetic algorithms such as Refined Soundex, NYSIIS, MetaPhone chosen from rockymadden string metric and Double Metaphone from Apache language codec library as specified in Section 4.2.3. In order to understand the blocking process in our application, let us consider an example as shown in Figure 4.2.

```
+---------+--------+--------+--------+
| id      | name   | city   | postcode|
|---------+--------+--------+--------|
| a1      | jmes   | magdeburg| 39106  |
| a2      | amily  | hamburg| 20098  |
| a3      | thomas | delhi  | 110002 |
| a4      | victokia| nwyork| 10003  |
| a5      | james  | magdeburg| 39104  |
| a6      | victoria| nwyork| 10003  |
+---------+--------+--------+--------+
```

Figure 4.2: Example of records containing duplicates

The DataFrame shown in the Figure 4.2 contains id, name, city and postcode columns. In order to generate the blocking key, the phonetic similarity measures are applied
directly on the columns. The blocking keys are trimmed by taking the sub-string of the obtained phonetic code. Let us consider the example in Figure 4.3, where BKVs are generated for name, city and postcode as dname, dcity and pc respectively. dname and dcity are generated by applying Double Metaphone algorithm and taking first two substrings from the generated Double Metaphone code. pc is generated by taking the substring of first two digits from postcode.

For combining the records with the same blocking key, we concatenated individual blocking keys generated for various attributes. For example, from the Figure 4.3 JM+MK+39 will be the concatenated blocking key for the record with id a1. In a particular case, the function for phonetic similarity is written so that it also handles null values. For example, If the value of the field is zero and when a phonetic algorithm is applied to it, the result obtained should be 0 not any phonetic code. Depending upon the variety and number of columns in the DataFrame, one or more blocking keys can used in this phase. After specifying the phonetic similarity techniques and selecting the blocking key, we registered the temporary table using Spark SQL. We used Double Metaphone technique as phonetic similarity to specify the blocking key. Registering a DataFrame as temporary table allows to apply SQL queries on DataFrame. SQL join statement with the blocking key is applied on the temporary table after converting the DataFrame, so that the records that have the same blocking key are combined as a same row in the table. The result obtained in this step will be in the form of DataFrame combined with records which satisfy the same blocking key values. From Figure 4.4, records (a1,a5) and (a4,a6) are compared with each other as the blocking key values (dname+dcity+pc) of the record pairs are equal and new column names name2, city2 and postcode2 are added to avoid confusion.
4.3.4 Pair-wise Comparison

Pair-wise comparison is the most expensive and time consuming step in the ER process. In the blocking step, records that have equivalent blocking key value are combined to one row in DataFrame. All the similar fields in the columns of the combined DataFrame are compared with each other in this step. If we want to compare all the columns with their similar column, each field in the column will be compared by its similar column once, making the computations complex. So, it is also required to select the columns apt for comparison to reduce computations.

Similar to blocking technique, User Defined Functions (UDFs) are used to define the field similarity metrics. withColumn() method from DataFrame API is chosen for this method to apply field similarity between two columns. Using withColumn() method, it is possible to specify required field similarity metric manually by the user. Then, the result of similarity value will be obtained by comparing two columns. The obtained similarity values will be in between 0 which is dissimilar and 1 means similar. If a field with zero is compared to a string in similar column, the resulting similarity value will be zero. For numerical columns, exact similarity metric, Hamming or Jaccard metric can be used for comparison. For non-numerical columns, Jaro distance, Jaro-Winkler or edit distance metric can be used. In our application we mainly used, exact metric for numerical columns and, Jaro and Winkler metric for non-numerical columns. The result of similarity is rounded to 2 decimal points. The result of this step will be in the form of a DataFrame with similarity values of compared records. In the example shown in Figure 4.5, Jaro and Winkler similarity metric is applied to the columns (name,name2), (city,city2) and (postcode,postcode2) resulting in the similarity values with the columns sim_name, sim_city and sim_postcode respectively.

```
+-----------------+-----------------+-----------------+-----------------+
| id  | id2  | sim_name | sim_city | sim_postcode |
+-----------------+-----------------+-----------------+-----------------+
| a1  | a5   | 0.94    | 0.98    | 0.92          |
| a1  | a6   | 0.95    | 0.95    | 1.0           |
+-----------------+-----------------+-----------------+-----------------+
```

Figure 4.5: Similarity values for attributes calculated using Jaro winkler similarity metrics

4.3.5 Classification

In the classification phase, labels are added to the records based on their field similarity values (features). The records are classified as match and non-match by adding labels to them. If the record is classified with the label 0, it represents a non-match, i.e. the records that are compared are not duplicates and label 1 represents a match of the records, which means the compared records are duplicates. In our application, it is possible to classify records using threshold-based classification or using Spark machine learning pipelines. In the following we introduce both classification methods.
Threshold-based classification We obtain a similarity value for each chosen attributes of record pairs from the pair-wise comparison step. Then in this classification method, we first sum up all the values and then create a new column named ‘total_sim’ to store it. The column, ‘total_sim’ is appended to the DataFrame. UDF for threshold ‘t’ is defined manually. If the value of field in ‘total_sim’ is less than the given threshold, the records are classified with label 0 and if value of ‘total_sim’ for record is equal or greater than the threshold ‘t’, then the record is classified with a label 1 as shown in Equation 4.1 and Equation 4.2.

\[
\begin{align*}
\text{total_sim} < t & \rightarrow \text{label} = 0 \\
\text{total_sim} \geq t & \rightarrow \text{label} = 1
\end{align*}
\]  

(4.1)  

(4.2)

Spark-ML-pipeline-based Classification In Spark ML pipelines, algorithms are classified into transformers and estimators. Transformer algorithms transform a given DataFrame into another DataFrame. Multiple transformers can be applied in a single ML pipeline. The output DataFrame of a transformer is taken by an estimator, which is also a ML algorithm, and derives a model from the DataFrame. In a ML pipeline, transformers and estimators are chained together [23]. Two methods fit() and transform() are used multiple times in ML pipeline. transform(DataFrame) method is used to produce a new DataFrame by transformer and fit(DataFrame) method is used to fit transformers in estimators. The output of the estimator is a transformer. In our application, we used various transformers and estimators in the ML pipeline, as illustrated in Figure 4.6. In the following, we introduce the ML pipeline in our application step by step.

- The training data contains features columns similar to that of the DataFrame obtained from pair-wise comparison with an additional column ‘label’. Label column contains the original labels from which the classifier will be trained.

- Transformer 1: The input DataFrame with similarity values in various columns is taken and all the features in those columns are combined into a single vector column by using the vector assembler transformer. The use of transformers after the using vector assembler step are optional steps. It depends upon the data we use to match the duplicates.

- Transformer 2: Applying String index model transformer on a column results in the transformed column with indexed labels and meta data added to it. It is mainly applied if the label column has String data in it.

- Transformer 3: Vector indexer model transformer is applied on the features column, which automatically identifies the categorical features from the feature column and transforms the original feature values to indexed values. Maximum categories are set to identify the distinct category values from features.
Machine learning classification model is trained by decision tree or random forest classifier by adding the label column and features column of the training data.

Transformer 4: Index to String transformer works contrary to String Indexer, it converts the indexed labels to original labels by giving input and output columns.

All the above steps form a ML pipeline and `fit()` is applied by specifying the train data. The training data is splitted into two parts such as 70% and 30%, first part is given to the classifier for training and the second part is used for testing the accuracy of the predicted labels of classifier using binary class evaluation.
metrics. If the accuracy of the trained classifier is not good (less than 80% in our application), it is possible to tune using cross validation methods.

- As the test data (result of pair-wise comparison step) does not have label column, a column is appended using `withColumn()` method to the DataFrame. After appending column to add labels in test data, the test data is added to the pipeline by `transform()` method. At last, the predicted labels are added to the ‘label’ column in test data and the quality of the predicted labels is calculated in evaluation stage.

4.3.6 Evaluation

The classified records are taken as input of the evaluation stage. Spark DataFrame method `count()` is used for calculating the total number of records in the DataFrame. Spark SQL query operations using WHERE clause are used to calculate number of true positives, true negatives, false negatives and false positives from which recall, precision and F-measure of the matches are calculated. The quality of the obtained results with varying datasets is discussed in Chapter 5.

4.4 Tuning

In order to get a higher performance, we tune the application with the following specific optimizations and settings:

Optimizations:

- The reusable data is stored in memory by using `cache()` method where ever the data is reused. For example, the training data is cached when using ML based classification and in evaluation step, the count of the total number of records is cached to make computations of TP, TN, FP, FN faster.

- `repartition()` method is used when data is loaded into Spark SQL and we specified the number of partitions according to the number of nodes that we use to run our application. This helps to distribute data equally among the workers in the cluster. Example: For 4 executors each with 2 cores in cluster, `repartition(8)` is used.

- According to the Spark official website, use of kryo serializer makes serialization faster than using default java serialization. Therefore, kryo serializer is used in our application for serializing objects instead of java serializer.

Settings:

- Several configuration settings such as default parallelism, maximum memory to be accessed by the executors and drivers are set while submitting the application to the cluster.
• Spark’s default Log4j property is set true in configuration properties, which enables logging of our application.
4.5 Summary

In this chapter, we started with Section 4.1, by discussing the reason why we chose to use Spark instead of MapReduce programming model. Next in Section 4.2, we explained the implementation choices made to develop the ER application in which the details of the programming language used, the development environment and the external libraries used are given. The workflow and methodologies used to implement the parallel ER application are discussed in Section 4.3. In this section, we explained how each step is implemented using Spark SQL and DataFrame API in detail. At last in Section 4.4, we explained the configuration settings and optimizations used to improve the performance.
5. Evaluation

In order to evaluate our Spark-based Entity Resolution (ER) application, we conducted various experiments. In this chapter, we introduce our experiments and discuss the results of various experiments to evaluate the performance of our parallel ER Application. In Section 5.1, we give detailed information on the setup of the experiment, where we discuss the test datasets used for evaluation, the cluster software and hardware. In Section 5.2, we describe the experiments conducted on our parallel ER application in order to check efficiency and effectiveness of the application. In Section 5.3, we evaluate and discuss the results of our experiments.

5.1 Experimental Setup

5.1.1 Input Datasets for Experiments

We used the data generated by external data generator named FEBRL data generator [14] to conduct the experiments. As it is difficult to acquire the gold standard data with varying amounts of size and duplicates, we made use of the external data generator. The FEBRL data generator was developed by Christian, which is a part of FEBRL project. FEBRL is an open source software available for record linkage process written in python. Using this generator, it is possible to generate the data with the required size and with varying amounts of duplicates. As illustrated in Table 5.1, 9 datasets were generated initially with the input size of $10^4$, $10^5$, and $10^6$ records each with 10%, 25%, and 50% duplicates. For testing scalability of the application, records with various sizes from $2*10^5$ to $8*10^5$ with 10% duplicates were also generated.

Dataset format: The generated data were saved in CSV format. The data contains `rec_id, given_name, surname, street_number, address_1, address_2, suburb, postcode, state, date_of_birth, phone_number, soc_id, blocking_number` as columns.
Training data: We need training data to train the required classification model for learning-based classification techniques. The records used for the training data were selected randomly from the generated data. To get the training data from the selected records, they were processed from preprocessing step to comparison step in ER process. The output of the comparison step was saved and labels were added in a manual way. We used training data with 35000 records for all experiments with different input datasets.

<table>
<thead>
<tr>
<th>Total number of records in the test dataset</th>
<th>Duplicates</th>
<th>Size in MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>(10^4)</td>
<td>1000 (10%)</td>
<td>1.12</td>
</tr>
<tr>
<td>(10^4)</td>
<td>2500 (25%)</td>
<td>1.12</td>
</tr>
<tr>
<td>(10^4)</td>
<td>5000 (50%)</td>
<td>1.12</td>
</tr>
<tr>
<td>(10^5)</td>
<td>10000 (10%)</td>
<td>11</td>
</tr>
<tr>
<td>(10^5)</td>
<td>25000 (25%)</td>
<td>11</td>
</tr>
<tr>
<td>(10^5)</td>
<td>50000 (50%)</td>
<td>11</td>
</tr>
<tr>
<td>(10^6)</td>
<td>100000 (10%)</td>
<td>113</td>
</tr>
<tr>
<td>(10^6)</td>
<td>250000 (25%)</td>
<td>113</td>
</tr>
<tr>
<td>(10^6)</td>
<td>500000 (50%)</td>
<td>113</td>
</tr>
</tbody>
</table>

Table 5.1: Test datasets with various size and amount of duplicates

5.1.2 Experiments Environments

The Spark default cluster that used for experiments contains 8 workers, in which one of the workers acted as the master node. In general, it is possible to run the application in Spark using its own standalone cluster or with the use of other cluster managers such as MESOS or Hadoop YARN. In this section we discuss more about the software and hardware environments of the cluster. At last, we discuss how the application is submitted to the cluster.

5.1.2.1 Software Environments of the Cluster

Ubuntu 16.04 version with the 64-bit operating system and Java 1.8 version were installed for all the workers. We used Spark version 1.6.1 with Hadoop 2.6.0 distribution. In order to establish a connection with the cluster, we used SSH secure shell and Open VPN. Spark standalone cluster is used for running the application without the use of cluster managers such as MESOS or Hadoop YARN. The Local file system was used for storing files in the cluster.
5.2. Experiments Design

<table>
<thead>
<tr>
<th>Cluster</th>
<th>Cores</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 worker</td>
<td>2</td>
<td>6 GB</td>
</tr>
<tr>
<td>2 workers</td>
<td>4</td>
<td>12 GB</td>
</tr>
<tr>
<td>3 workers</td>
<td>6</td>
<td>18 GB</td>
</tr>
<tr>
<td>4 workers</td>
<td>8</td>
<td>24 GB</td>
</tr>
<tr>
<td>5 workers</td>
<td>10</td>
<td>30 GB</td>
</tr>
<tr>
<td>6 workers</td>
<td>12</td>
<td>36 GB</td>
</tr>
<tr>
<td>7 workers</td>
<td>14</td>
<td>42 GB</td>
</tr>
<tr>
<td>8 workers</td>
<td>16</td>
<td>48 GB</td>
</tr>
</tbody>
</table>

Table 5.2: Configuration of cluster used in experiments

5.1.2.2 Hardware Environments of the Cluster

Each worker has 2 CPU cores with 6 GB main memory. Depending upon the experiment, the cluster configuration was modified so that the required number of workers were accessed. In Table 5.2, the different cluster configurations used for the experiments in Section 5.3 are displayed. The workers in the cluster were connected in the form of a bus topology. Although the workers contain similar software in them as discussed in Section 5.1.2.1, the hardware of the workers is different from one another. This type of configuration is called heterogeneous configuration. For instance, among two workers, one worker has Intel Xeon E5-2650 and the other has Intel Xeon E5620. We did not chose this kind of heterogeneous configuration on purpose and we had no other choice than to use it.

5.1.2.3 Submitting Application to the Cluster

Our application was shipped to jar file and it was submitted to the cluster by launching in `spark-submit`. Generally using spark-submit, various configuration settings can be specified while launching the application and it is also possible to run the application in cluster or in local mode [23]. We chose to run the application in cluster for our experiments. Depending upon the experiment, the class in which application has to be run, the maximum number of workers to be used, the maximum number of cores, and the input files were altered and specified via spark submit.

5.2 Experiments Design

Various observations were made using the generated datasets and with the Spark standalone cluster. Varying the number of workers used in the cluster, different experiments were carried out. We divide the experiments into three types according to the cluster setup and the datasets used. In this section, we discuss design of the experiments and
the reasons behind the experiments performed. The experiments design is shown in the Table 5.3.

**Experiments for Evaluating Blocking**

The experiments are performed by using 8 workers in the cluster and the datasets $10^4$ to $10^6$, each with duplicates of 10%, 25%, and 50% which are shown in Table 5.1. By taking the same input datasets and nodes, experiments for evaluating blocking and learning-Based classification were performed.

To know the performance of the blocking step in the application, we evaluated the blocking step. The design details are displayed in first row of Table 5.3. For calculating the performance of blocking, we calculated the search space i.e, the number of records compared with each other after blocking. Next, in order to know the quality of the blocking, true matches of the records which are compared after blocking in the comparison step are calculated. In Section 5.3.1, results of blocking experiments are discussed in detail.

**Experiments for Evaluating Learning-Based Classification**

To evaluate the accuracy benefit using learning-based classification other than simple threshold-based classification and applied blocking method, we designed the experiments, whose designing details are showed in the second row of Table 5.3. This experiment was first evaluated on observing the runtime of ER process with varying classification steps. Last, in this experiment we evaluated the quality of the results obtained in the classification step. The results of these experiments are discussed in detail in Section 5.3.2.

**Experiments for Evaluating Speed-up and Efficiency**

With the same input, the parallelism of the threshold-based classification and machine learning based classification methods were similar. So, we have used threshold-based classification method to perform these experiments instead of the decision tree and random forest. This is because we require the higher size of data to perform experiments in parallel and high amounts of training data is needed for machine learning based classification methods.

As we know, our application is implemented to run in the parallel environment. In order to check the parallelism of the application, we designed these experiments. Speed-up and efficiency are calculated to check the parallelism of the application. The design of the experiments are showed in the third row of Table 5.3. The workers in cluster were varied from 2 to 8 and the dataset with $10^6$ records and 10% duplicates was taken. In order to evaluate our application for relatively large-volume data, we took the dataset with $10^6$ records as input. The results of these experiments are discussed in Section 5.3.3.
5.3. Experiments Results, Evaluation and Discussion

Experiments for Evaluating Scale-out

To check the scalability of the application, scale-out experiment was performed by increasing the amount of workers and datasets. Scale-out experiment was performed instead of scale-up, as we are increasing the number of workers in the cluster without improving the hardware of the machine. Design details were shown in the fourth row of the Table 5.3. For calculating scale-out, we increased the number of workers from 2 to 8 with increasing the size of the data from $10^5$ to $8 \times 10^5$ respectively but in the same rate. The results of scale-out experiment are discussed in Section 5.3.4.

5.3 Experiments Results, Evaluation and Discussion

In this section, we discuss the results of the experiments that we discussed in the previous section. In all the observations, the application was executed 5 times randomly and the average time was taken for the robustness of results.

5.3.1 Experiments for Evaluating the Blocking

The search space and the quality of true matches after blocking were evaluated in this experiment. Standard blocking is used for indexing step in our application. As discussed in previous chapters, choosing a blocking key is crucial for this step. The search space and runtime of the application also depend upon the blocking key. Also, the number of true matches taken for comparison are dependent on blocking key.

The blocking key value chosen for this experiment was first two characters of Double Metaphone of Surname + first two characters of Double Metaphone of given name + two digits of the postal code.

Blocking key and search space in the application are dependent with each other. If there are more fields specified in the blocking key, the search space reduces which may also reduce the comparison of true matches. Some of the true matches (duplicates) in the datasets which does not satisfy the blocking key will not be taken to the comparison step.

Search space

In this experiment, the amount of records in the search space was calculated. Search space represents the number of records which are compared in pairwise comparison step after the indexing step. Generally, without blocking the search space for $n$ records will be $O(n^2)$. The reduction in the search space for our application was compared to that of the original search space without blocking for various datasets as shown in figures Figure 5.1, Figure 5.2 and Figure 5.3. The search space of the records without blocking for $10^4$ datasets is $10^4 \times 10^4$. By applying blocking technique the search was reduced to 5000 for 10% duplicates and 7000 for 50% duplicates as shown in Figure 5.1. For $10^4$ records, the search space was less than the input and search space of records with 50% duplicates was more with that of 10% and 25% duplicates. For input of $10^5$ and $10^6$ records, the search space was 4 times of the input records and the search space was more for the records with 10% duplicates.
<table>
<thead>
<tr>
<th>Experiments</th>
<th>Evaluation goals</th>
<th>Number of nodes</th>
<th>Evaluation metrics</th>
<th>Design goals</th>
<th>Type</th>
<th>Experiments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evaluating the blocking step</td>
<td>Check performance and quality of blocking step</td>
<td>10, 4, 8, 12, and 16</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Evaluating Learning-based classification</td>
<td>Evaluate the accuracy benefit using Learning-based classification other than Threshold-based classification</td>
<td>10 datasets, each with 10% duplicates</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Evaluating speed-up and efficiency</td>
<td>To calculate parallel performance of ER application</td>
<td>2 to 8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Evaluating scale-out</td>
<td>To check scalability of ER application</td>
<td>10^5, 2<em>10^5, 3</em>10^5, 4<em>10^5, 5</em>10^5, 6<em>10^5, 7</em>10^5, and 8*10^5, each with 10% duplicates</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 5.3: Experiments design**
5.3. Experiments Results, Evaluation and Discussion

Figure 5.1: Reduction in the search space for $10^4$ datasets

Figure 5.2: Reduction in the search space for $10^5$ datasets

Figure 5.3: Reduction in the search space for $10^6$ datasets

Quality compared to the situation without blocking

If there is no blocking step, all the records will be compared with each other in the comparison step which makes it to compare all the true matches in comparison step. As we use blocking technique, only the records which satisfy the blocking key were compared to each other and few true matches were left out from comparison step. This generally happens when the true matches does not satisfy the blocking key. As discussed before, all the true matches will not be taken to the comparison step. With the blocking key specified, the number of true matches which were compared in the comparison step range between 70% to 80% for various datasets and duplicates. The number of true matches can be observed from Figure 5.4, Figure 5.5 and Figure 5.6.
5.3.2 Experiments for Evaluating Learning-Based Classification

Runtime of Threshold-Based and Learning-Based Classification

Runtime of Threshold-based and Learning-Based classification was calculated for the entire ER process. The ER process starts from loading the data for the preprocessing step and ends with the analysis of results in the evaluation step. The process does not save results to the text file. Only the classification step was varied by decision tree, random forest, threshold-based, and the runtime were calculated accordingly. It was observed that there was very small difference in the runtime by varying the classification step with the decision tree and random forest. There was decrease in the runtime for threshold based classification, as it does not learn from the training data. Figure 5.7 shows the runtime of the ER process with decision tree, random forest and threshold-
based in the Spark default cluster with 8-workers. It was also observed that the runtime of the dataset does not depend on the number of duplicates in it as there was no much difference from the runtime of the same dataset with varying the amount of duplicates in it. For ML algorithms, we observed maximum of 0.5 minutes difference in runtime of data with $10^4$ and $10^5$ records. Average time taken to run ER process with $10^4$ datasets was 1 minute, with $10^5$ datasets was 1.5 minutes for all duplicates. Average runtime for $10^6$ records with distinct duplicates was 5 minutes for ML algorithms.

Figure 5.7: Runtime of ER application with different classification methods

Quality of Threshold-Based and Learning-Based Classification

The quality of the results obtained by the classification algorithm were evaluated in this experiment. The quality was compared between random forest, decision tree and threshold based algorithms. We performed this experiment by obtaining the results as true positives, true negatives, false positives and false negatives from the evaluation step. Accordingly, Precision, Recall and F-measure were calculated for datasets of $10^4$ and $10^5$ records with 10%, 25%, and 50% duplicates. We provided the training data for the classifier so that the algorithm learns from the training data. As we know, the higher and diverse the training data, we get high precision and recall values. The results of precision are visualized in Figure 5.8 and Figure 5.9, they were based on the same amount of given training data given to $10^4$ and $10^5$ records. The F-measure values for the records are shown in Figure 5.10 and Figure 5.11. The recall value attained for all the classifications was above 0.95, so the quality of the results in F-measure is increased when compared to the precision values. We can observe the difference between the quality of the results obtained from decision tree classification and random forest classification from the figures. For our test data, values of precision and F-measure for classification using random forest were higher than that of the values of classification using decision tree algorithm. The accuracy of the classifier seemed better for the records
5. Evaluation

Figure 5.8: Precision for $10^4$ records with various classification methods

Figure 5.9: Precision for $10^5$ records with various classification methods

Figure 5.10: F-measure for $10^4$ records with various classification methods

Figure 5.11: F-measure for $10^5$ records with various classification methods

with more number of duplicates. As we provided same training data to $10^4$ and $10^5$ records, there was decrease in the quality of the results for $10^5$ records, when compared to the quality of $10^4$ records. For the data containing $10^6$ records, training data have to be increased and improved to obtain better accuracy. Due to the limited time period, training data was not increased for evaluating $10^5$ and $10^6$ records.

5.3.3 Experiments for Evaluating Speed-up and Efficiency

We have analyzed the runtime, speed-up, and efficiency by regulating the number of workers in the Spark cluster.

Runtime with varying workers in cluster

Total runtime of ER process with our parallel ER application was calculated by varying the number of workers from 2 to 8. The cluster configurations of these workers can be
observed in Table 5.2. With one worker, time taken was 22 minutes. It took about 11 minutes for 2 workers and runtime was reduced to 3.9 minutes when we used 8 workers in the cluster. The runtime on increasing the number of workers can be observed in Figure 5.12.

![Figure 5.12: Runtime of data with $10^6$ records and 10% duplicates with varying amounts of workers using threshold-based classification](image)

**Speed-up and efficiency**

Speed-up and efficiency were calculated based on the runtime in Section 5.3.3. As it is hard to acquire sequential algorithm for calculating the speed up, we took the total runtime for ER process in one executor and then calculated speed-up. The speed-up and efficiency can be illustrated from Figure 5.13 and Figure 5.14.

### 5.3.4 Experiments for Evaluating Scale-out

The total runtime of the ER process were calculated as illustrated in the Figure 5.15. Ideal performance in scale out is achieved when there is no increase in the runtime, with increasing amount of data and workers [51]. But, in reality it is not possible to achieve due to various possibilities which are discussed in Section 5.3.4.

**Factors that lower the performance**

In this section, we discuss factors that were affecting the performance of the parallelism of our parallel ER application. From Section 5.3.3 and Section 5.3.4, we can observe that the results were not similar to that of the ideal results. In reality, it is hard to achieve the ideal performance in parallelism due to various factors [51]. As we can see from Figure 5.12, Figure 5.13 and Figure 5.14, the actual evaluation result is still far from the perfect, which is indicated by the distance between the ideal runtime, speed-up,
efficiency and the actual ones. For example, when we use 8 nodes in the cluster, the speed-up ideally can be 8 times, however, the actual speed-up is only 5 times faster. The ideal efficiency should be always equal to 1 when we increase the number of nodes used in the cluster. But, the actual efficiency is lower than 1, and it is around 0.6 when we use 8 nodes used in the cluster. We analyze the following reasons for the performance loss:

1. As we discussed in Section 5.1.2.2, the use of heterogeneous clusters causes variations in the parallelism of the application [62].

2. Even though the data is distributed evenly among the clusters, we observed few executors were performing the tasks slowly compared to the others.

3. The network in which the workers were connected are in the form of bus topology, which increase in the communication costs between the workers with increasing amounts of nodes in the cluster.
4. The files were accessed locally from the clusters, without the use of Distributed File System and we have used external libraries in the application for string similarity metrics which have caused additional computational overhead in the cluster.
5.4 Summary

In this chapter, we provided the details of the experiments, that are performed in order to evaluate our parallel ER application. We started the first section by discussing the experimental setup, where we discussed regarding the input datasets, the software and hardware environments of experiments. In the second section, we discussed the design of the experiments, where we divided the experiments into three types based on the input datasets and nodes. At last, in the third section, we discussed the results of the experiments in detail.
6. Conclusion and Future Work

6.1 Conclusion

We designed, implemented and evaluated a parallel Entity Resolution (ER) application using Apache Spark in this thesis. A complete ER application is designed with general ER processes containing pre-processing, blocking, pair-wise comparison, classification and evaluation steps. In order to run the application in parallel environments, we implemented the application using Apache Spark. Our application was mainly implemented using Spark SQL and ML components. To increase the performance of the application, Spark SQL component was used which supports DataFrame API. Mainly, the catalyst optimizer in Spark SQL makes optimized execution of queries which helps in increasing the performance of the application. Using the concept of set-similarity joins, the blocking technique was implemented using a blocking key. User Defined Functions are used to define the phonetic and similarity based string metrics. Spark-based ML pipelines are used for learning based classification methods in our application. To attain maximum performance, various tuning techniques are applied while implementing the application. Most of the computations are performed in-memory which speeds up the ER process.

In order to evaluate the effectiveness and efficiency of the application, various experiments were designed and conducted. The datasets of various sizes and duplicates are generated from FEBRL data generator. It is observed that with use of an optimal blocking key, it was possible to reduce search space and increase the true matches in comparison up to 80%. We compared the runtime and quality of various classification methods and observed the threshold-based attains less runtime. Use of learning-based ML algorithms achieves accurate result quality when the sufficient training data was made available.

As our application is implemented to run in parallel environment, the parallel performance of the application was calculated. Total runtime of the application was calculated by varying the number of workers from 2 to 8. We calculated speed-up and efficiency of the application depending on the runtime obtained. By increasing the same amount of
datasets and the workers, scale-out was calculated. As the number of workers increased in the cluster, the parallel performance of our application was decreased. The reason was mainly due to the communication overhead caused and the use of heterogeneous workers in the cluster. The other directions where our work can be extended are discussed in the following section.

6.2 Future Work

Although various techniques and experiments are performed in our thesis, we came across various other possibilities where our work can be extended. In this section, we provide possible future insights where our work can be extended and improved. The following points discuss the future research directions of our work.

**Comparison with MapReduce:** Our work can be extended by comparing our parallel ER application with the MapReduce model. Even though our implementation is different from the implementation of the MapReduce model, comparison of the results can be made by using the same real world datasets which are used in calculating the performance of MapReduce model.

**Additional Algorithms:** Blocking techniques can be extended by implementing various other blocking algorithms than using only standard blocking to increase the quality of the results in blocking step. Algorithms like sorted neighborhood, and use of multi-pass blocking techniques can be implemented. We have evaluated our application using only few similarity metrics, exact performance of the application can be measured when various similarity measures are applied and evaluated.

**Web UI:** An interesting future work would be developing an online web user interface, which has various options to choose from various algorithms in blocking and classification techniques, similarity measures. With this, the user can be able to perform parallel ER, by selecting the desired techniques.

**Advanced tuning techniques:** In our work, we used few tuning techniques to increase the performance of the application. Advanced tuning techniques can be performed on the application which may further result in maximizing the performance of the parallelism of the application. For instance, using advanced kryo serialization options, changing the configuration settings etc.
Bibliography


[27] Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques*. Elsevier, 2011. (cited on Page 15)


[53] Chen Wang and Sarvnaz Karimi. Parallel duplicate detection in adverse drug reaction databases with spark. (cited on Page 30)


Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Magdeburg, den 07.11.2016