WALAILAK JOURNAL
Applied Sciences
http://wjst.wu.ac.th https://doi.org/10.48048/wjst.2021.7221

Comparative Study of Record Linkage Approaches for Big Data

Randa MOHAMED[1,*], Ali EL-BASTAWISSY[2], Eman NASR[3] and Mervat GHEITH[1]

[1]*Computer Science, Faculty of Graduate Studies for Statistical Research, Giza, Egypt*
[2]*Computer Science, MSA University, Egypt*
[3]*Independent Research, Egypt*

([*]Corresponding author's e-mail: randa_mohamed_cs@yahoo.com)

Received: 19 July 2019, Revised: 19 January 2020, Accepted: 19 February 2020

Abstract

Record linkage is a challenging task for Big Data. This paper, hence, attempts to shed light on record linkage approaches for Big Data by comparing three dimensions involving record linkage phases, dataset properties, and parallel processing approach for Big Data. The current state of art have only conducted comparative studies between record linkage approaches. There has been only one comparative study exploring the whole record linkage framework of the relational database. It is believed that the focus of the present study on the dimensions of the parallel processing approaches for Big Data and dataset properties was worth exploring. It was found that first, data exploration was almost a non-existing phase despite its importance of exploring the dataset being examined; second, techniques that handle data standardization and preparation phase of the first dimension were not extensively covered in the literature which can directly affect the results' quality; third, the record linkage in unstructured data was not yet explored in literature; fourth, the MapReduce was used in about 50 % of the selected studies to handle the parallel processing of Big Data, but due to its limitations, more recent and efficient approaches had been used, such as Apache Spark and Apache Flink. Apache Spark is just recently adapted to resolve duplicates due to its supporting of in-memory computation, which makes the whole linkage process more efficient. Although the comparative study includes many recent studies supporting Apache Spark, adopting Apache Spark to solve the problem of record linkage is not yet well explored in literature, as more researches need to be conducted. In addition, Apache Flink is still rarely used to solve the record linkage problem of Big Data. Fifth, pruning techniques, used to eliminate unnecessary comparisons, are not adequately applied in the covered studies despite their effect on reducing the search space resulting in a more effective Record Linkage process.

Keywords: Big Data, Flink, Record linkage, Hadoop, MapReduce, Spark

Introduction

Big Data are not about how much the size of data is increasing, but rather aboutthe situations where the size, the variety, and the velocity of the data exceed an organization's capability to process such an amount of data [1]. In such situations, data processing software are usually inadequate to handle it, and there must be alternative ways to process Big Data to gain value from it [2]. Since the data size is growing extremely fast, taking the right decision on time becomes very time-consuming, especially if the data comes from different sources with different formats. There are many challenges involved in Big Data such as capturing, storing, analyzing, querying, processing, and visualizing data [3]. The size of data (volume) is not the only factor that characterizes Big Data, but there are many other factors; the most famous five are volume, variety, veracity, velocity, and value, known as 5 vs. of Big Data [1]. Real-world data are dirty and contain many potential errors, happening due to incomplete, noise, and inconsistent

values of the data. Data cleaning is the process of detecting and resolving corrupted, missing, and inconsistencies from the dataset. Record linkage is an important issue in data cleaning. Record linkage in literature has many equivalent names such as entity matching, entity resolution, and duplicate detection. Record linkage is one of the key challenges of data integration, which is the process of matching records that refer to the same real-world entity [4]. Lack of data standardization, typographical errors, changing demographical data and data integration are all reasons for duplicates in the datasets. Preparing and cleaning data for analytics purposes are very important tasks because they affect the quality of the obtained results; they are considered the hardest and most time-consuming tasks, taking approximately more than 80 % of the whole time spent in the analytics process.

According to Gartner, $44 billion in 2014 alone were spent on preparing data from many sources for use in data analysis [5]. Integrating data from many sources is critical for maximizing the capability of data-driven decision-making. Data integrated from multiple sources can drastically increase the power of retrieved information and provide the opportunity to answer questions that are impossible to be solved with a single data source. As an example, the analysis of health data integrated from multiple sources like electronic health records, drug and toxicology databases, genomics, and social media environments, is a key driver to advanced precision medicine [4].

According to Dong *et al.* [6], adding the dimensions of Big Data makes record linkage more challenging and time-consuming process due to many reasons such as:

1. The increasing volume of Data that needs efficient parallelization techniques to process it
2. The heterogeneity of data structure as data comes from different sources with different formats
3. The dynamic and continuously evolving data
4. The data comes from multiple applications with different accuracy requirements. All the above challenges call for new approaches for handling Big Data.

The main objective of this paper is to conduct a comparative study on the relevant available literature about record linkage approaches for Big Data. The structure of the comparative study consists of three dimensions: the first dimension is about record linkage phases, the second dimension is about the used dataset properties, and the third dimension is about parallel processing approaches for Big Data.

The rest of this paper is organized as follows. Section 2 discusses the proposed dimensions of the comparative study of record linkage for Big Data. Section 3 presents a classification of the covered studies according to parallel processing approaches for Big Data. In section 4, the results of the comparative study are discussed. Finally, the conclusion and future work are given in section 5.

Proposed dimensions of the comparative study

Comparative studies of record linkage approaches for Big Data are still insufficiently conducted. The available literature has only published comparative studies about record linkage approaches or techniques for relational databases. There has been only one comparative study focusing on record linkage frameworks of the relational database, thereby encouraging the present study to perform a comparative study about record linkage approaches for big data.

Kopcke *et al.* [7] comparatively analyzed eleven frameworks of entity matching. Their proposed framework utilized both training and non-training data, as it compares three frameworks with no training data and 8 frameworks with training data. They suggested two types of comparison criteria: criteria for functional comparison and criteria for evaluation comparison. The criteria for functional comparison included entity type, blocking methods, training selection, matcher, and the combination of matchers. The criteria for evaluation's comparison included the type of problem, the applied matching strategy, and the achieved effectiveness and efficiency performance.

In the present study, record linkage approaches has been extended to include parallel processing approaches for Big Data, aside from adding many unaddressed comparison features, which will be explained in the next subsections. The proposed comparative study could be exploited as a guide or framework for a more efficient and effective record linkage process of big data or even relational datasets. As depicted in **Figure 1**, the proposed comparative study consists of three dimensions.

The first is about record linkage phases, the second is about the used dataset, and the third discusses the used approaches for processing Big Data and Very Large-Scale datasets. All these approaches are centralized around the parallel distribution of the large dataset. Next, the three dimensions are detailed.

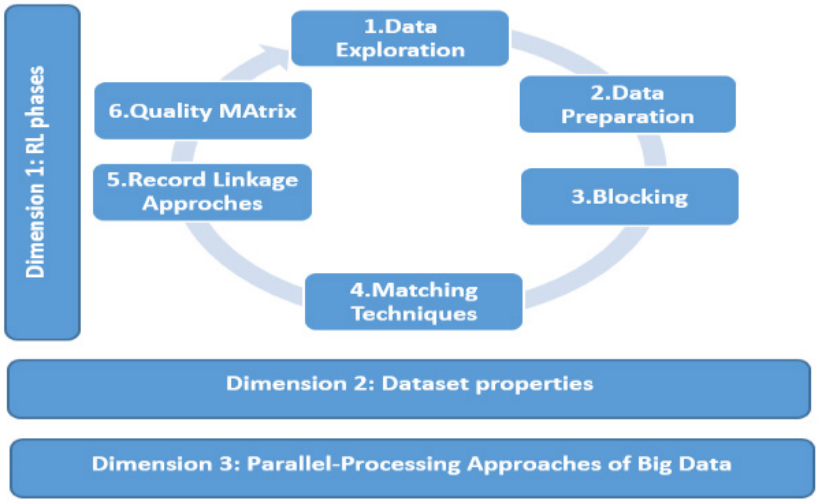

Figure 1 The proposed dimensions of the comparative study of record linkage for Big Data.

Dimension 1: The record Linkage phases

As depicted in **Figure 1**, the traditional record linkage process runs through 6 steps as shown in the first dimension. Record linkage process begins with data exploration, aimed at investigating the dataset that will be analyzed and comprehended well. The second step is data preparation in which the dataset is prepared to be ready for data cleaning or data integration. In this step, the features of record linkage are chosen, and some data transformation and standardization are done. The third step is blocking which aims to reduce the search space of the record linkage process and hence increase the efficiency, as the time of performing the record linkage will be reduced. In the blocking step, the data is distributed to a number of blocks in which the elements that only reside in the same block are compared. The fourth step is selecting similarity detection techniques. In this step, the elements are compared by using one or more of the similarities techniques. The fifth step is selecting the record linkage approach for Big Data in which the whole record linkage process among entities is done using one or a mixed number of approaches. Many approaches can be applied such as probabilistic approaches, machine-learning approaches, distance-based approaches, rule-based approaches, and graph-based approaches. The final step tests the accuracy of the selected approach for record linkage by the quality matrix measures. The next subsections explain these 6 phases in detail.

Data exploration

Data exploration is the analysis to understand such characteristics as data size, data completeness, the collaboration between attributes, and data outliers. It aims to describe the data by using statistical and visualization techniques. Data exploration brings an important aspect of data for further analysis. There are many statistical functions used to analyze the data such as count, max, min, mean, median, mod, variant, standard deviation, skewness, correlation, and charts. R, SPSS, Python, and tableau are the best-known data visualization and analytical tools used for data analysis.

Data preparation

It means manipulating the data to be ready for analysis. Taking the data as it is will not guarantee to have an accurate model and hence accurate results. Data preparation has many steps, which are:

Feature selection: it studies how to select the best list of attributes that will be used to construct the model. It includes reducing the dimensionality, removing redundant features, reducing the amount of data needed for algorithm learning, and improve algorithm accuracy [8]. Feature selection has the following advantages:

1. It reduces the time taken to learn by Machine Learning algorithms.
2. It reduces complexity.
3. It makes the model easier to interpret.
4. It improves the accuracy of the model if the right subset is chosen.

- Handling null values: null values may be predicted using many methods as taking the average values, taking the values of similar objects attributes or simply omitting the whole record.
- Normalization or data transformation: It is the technique that transforms a string to a common and smaller set of string values. Usually, normalization is accomplished in 2 phases: (1) finding a pattern in the string using Regular Expression (RE), and (2) replacing one pattern by another [5]
- Standardization: It is the process of implementing a certain standard or format to the data (e.g. Date, time, name, and title) [5].

Blocking

Traditional record linkage approaches apply matching techniques on a Cartesian product of n inputs entities. These approaches result in a complexity of O(n2) which causes enormous execution time for the big dataset. Blocking techniques have been proposed to reduce such execution time by only comparing the elements that are more likely to be matched. Blocking is done by partitioning the data into smaller blocks to process them in parallel where records exist in the same block are only compared. Blocks are chosen in a way such that records of the same block are more likely to be similar and records rely outside that block are dissimilar. Blocking is done using blocking techniques such as choosing the initial 3 characters for the first name of the employee. There are other different blocking methods for record linkage such as Standard Blocking, Sorted Neighborhood, Q-gram Indexing, and Canopy Clustering with TF-IDF [9].

Selecting matching techniques

The purpose is to identify the similarities between the values of the fields used in the record linkage process. Matching is done by using one or more methods of record linkage, which includes:

- Character distance similarity-based: Distance functions map 2 pairs of strings to a real number r, where the smaller value of r indicates greater similarity between the two strings [10].
- Token similarity-based: Character-based similarity's methods work well for typographical errors but not well in case of words re-arrangements in the same sentence like computer science faculty vs. faculty of computer science. In such cases, token-based is a good choice [10].
- Phonetic similarity-based: Character or token-based Similarity methods only focus on detecting string similarity between items but fail to detect similar phonetic items. In such cases, phonetic similarity methods are a good choice. They are measures based on phonetic [11].
- Numeric similarity-based: Aims to detect similarity between numeric fields. The same methods described above are used in this case. A simple query to detect the exact number or range of numbers can be used to detect similar numeric items.

Selecting the suitable record linkage approach for Big Data

Record linkage approaches are used to detect similarity between database records that consist of multiple fields. These approaches can be classified into 5 categories as summarized below:

1. Probabilistic approaches: Probabilistic approaches use likelihood ratio or more likely decision and statistical theory to establish accurate links between records of typographical errors. Probabilistic approaches assign a percentage to indicate the matching probability [12].
2. Machine learning-based approaches (ML): record linkage can be viewed as a pattern classification problem, which aims to classify a pattern into a finite number of classes. Similarly, the goal of record linkage is to classify the status of pairs of records into match or non-match. In particular, given a set of labeled classes, machine-learning algorithms build a model that can be used to predict the class of new unlabeled data [13].
3. Distance-based approaches: Distance-based approaches compute the distance between records in two datasets. In general, for each record in file "A", a distance between it and every record in file "B" is computed, then the nearest distance and second nearest distance in file "B" are considered [14].
4. Rule-based approaches (deterministic matching approaches): Deterministic approaches use combinations of algorithms and business rules to determine when 2 records are matched. Deterministic approaches can catch simple errors like typos, transpositions, and phonetic variations [12].
5. Graph-based approaches: The graph is constructed of the chosen attributes. Matched records of the graph will be linked together. The nodes of the graph represent the attributes. Edge is used to connect two nodes labeled by the similarity between them. The graph-based approach can achieve high linkage quality at the cost of higher computational complexity.

Measuring the quality of record linkage (Quality matrix)

According to Köpcke *et al.* [7], there are many requirements that any record linkage approach should fulfill. (1) Effectiveness: The main goal of record linkage approaches is to achieve high-quality results w.r.t precision and recall, which means the results of it, should only include elements that refer to the same real-world entity and no other elements. (2) Efficiency: record linkage systems should retrieve results in a fast manner even for a large or big dataset by applying some blocking approaches. (3) Generality: Entity resolution methods should be applicable in many matching tasks in various application domains and for the different data models. As Yousef illustrates in [15], the quality of record linkage approaches is measured by the confusion matrix, which is used to compare actual matched records with non-actual matched records. These measures include True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN).

Dimension 2: Dataset properties

It represents the properties of the used dataset. Two properties are used: the first one is the type of the used dataset, and the second one is the size of the used dataset. The type of the used dataset may be structured, semi-structured or unstructured while the size of the used dataset may be a very large-scale dataset or Big Data. Structured data concerns all the data that can be stored in relational DB. It is very simple to enter, store or query. As the sources of data are growing very fast, structured data is only about 10 % of the available data. Semi-structured data does not reside in a structured manner but it has some organizational properties. CSV, XML, and JSON files are examples of semi-structured data. Unstructured data represents the majority of Big Data as it represents about 80 % of the data and it cannot comply with any structure. It is everywhere and often includes images, text, videos, and web pages. Unstructured data is either human-generated or machine-generated. Satellite images, scientific data, photographs, videos, and radar or sonar data are all examples of machine data while social media data, mobile data and website contents represent human-generated data. Context analysis is used to organize a large amount of textual data into a standardized format. It could be carried out in two ways: quantitatively by counting the words or qualitatively by coding. The quantitative method is done by counting the frequency of certain keywords while the qualitative method is done by identifying similar themes or concepts in the dataset.

Dimension 3: Parallel processing approaches for Big Data

Preparing and cleaning data for an analytic process is a very important step to guarantee an accurate result because of the concept of garbage in garbage out. Actually, it is the most exhausting and time-consuming process as it takes more than 80 % of the time dedicated to the analytic process. 44 billion

dollars were spent in 2014 to prepare the data integrated from many sources for the analytic process according to Gartner [5]. Record linkage is not an easy or trivial process as it takes many hours or even days if the size of the data is increasing [3]. In such cases, traditional record linkage will not be efficient nor effective hence, new approaches have been proposed to address the challenges of record linkage for Big Data. These approaches could be categorized into 4 main categories: MapReduce based, Spark-based, Flink based and other approaches. All three approaches are depending on partitioning the big dataset into a number of blocks and processing them in parallel according to the techniques of each approach. Next is a brief description of these approaches followed by a comparison between them.

MapReduce-based approach

MapReduce is a shared-nothing architecture specially designed to process the exhaustive workload. MapReduce depends on parallelly distribute the workload between many numbers of nodes in a way that guarantees fault-tolerant [16]. The main advantage of MapReduce programming model is its cost-efficiency as it relies on commodity hardware, so it is not expensive to scale out at reasonable prices. MapReduce is designed to detect and handle failures at the application layer, as it provides fault tolerance and high availability. In MapReduce paradigm, the input data is partitioned into smaller chunks or blocks and each one is processed separately. The data in each block is presented in the form of a key-value pair.

The processing is done where the data is located using 2 main functions, Map and Reduce. The Map function is responsible for generating an intermediate result in the form of key-value pairs according to the required query. The reduce function aims to generate the result by collecting the output of each mapper and produce one final output [16]. MapReduce can be utilized to solve the record linkage problem for Big Data. The Map function will be responsible for reading the input data in parallel after partition them into small chunks in key-value pairs (blocking key, entity). The default hash function is responsible for redistributing the blocks using their blocking key to the reduce functions. The reduce function is responsible for running the record linkage techniques in parallel using one or combinations of similarity techniques where blocks have the same blocking keys will be processed by the same reducer as illustrated in the pseudocode depicted in **Figure 2** [9].

```
Map()
{
    Foreach input entity
    {
        Determine the blocking key
    }
    Return Key-value pairs (blocking key, entity)
}
Use the default hash partition to assign each block to the Reduce()
Foreach Block
{
    Reduce ()
    {
        Calculate Similarity between entities()
        Return matching entity pairs
    }
}
```

Figure 2 Pseudocode of record linkage for Big Data using basic MapReduce model [9].

Spark-based approach

Apache Spark is a fast clustering computing technology built on top of the MapReduce framework. It extends the functionality of the MapReduce model to efficiently handle big data streams, as it is an in-memory clustering computation. Handling data streams in memory helps to increase the processing speed of the applications, as there is no time spent on moving the data to and from disk. Spark provides an API

for many programs as Java, Scala, and Python. Spark provides a layer for a large number of tools such as Mlib for machine learning, Structure data processing, Spark SQL, GraphX for graph processing, and Spark Streaming for stream processing [17].

Spark is a framework based on the design of Resilient Distributed Dataset (RDD). There are 2 types of RDD operations: transformations and actions. Transformation is a function, which produces a new RDD from the existing one. There are many functions of transformations such as map(), unions(), distinct(), groupby(), key(), sortbykey() and join(). Actions are following transformations and return results to the driver program. There are many functions of actions as collect(), Count(), reduce(), take(), CountByValue(), foreach(), and aggregate(). Transformations are lazy executions that mean they should be followed by actions to take place. In-memory computation is the main advantage of Spark, which means intermediate data is kept in memory, therefore it will allow 100 times faster computations than the Hadoop MapReduce paradigm [18].

Apache Flink-based approach

Apache Flink is an open-source stream processing framework developed by Apache software foundation. It is a distributed streaming engine written in Java, Scala, Python, and SQL. Flink executes the program in a parallel manner. It also enables the execution of batch and streaming processing. Also, it provides a high throughput, low latency streaming engine. Flink is used for executing the data as a pipelined fault-tolerant data flow [19].

Flink architecture has four main layers, which are deployment, core, APIs, and libraries. The core of Flink is a distributed data flow engine, which is responsible for executing data flow programs. Flink has two types of APIs, the first one is dataset API which is used for processing finite datasets (batch processing) and the second one is data stream API that is used for processing potential unbounded data streams (stream processing). Flink core runtime is a streaming data flow engine and both dataset and data stream APIs create runtime programs executable by engines on top of the core APIs. Flink has libraries and APIs that generate datasets and data stream API programs. Currently, there are many specific libraries such as Flink machine learning for machine learning, Gelly for graph processing, and table for SQL-like operations. Flink cluster includes three types of processes: the client, job manager, and track manager. Clients take the program code, transform it into a data flow graph, and then submit it to the job manager. The transformation phase examines the schema of data exchanged between operators and creates serializer and other schemas specific code. Dataset programs go through a query optimization phase similar to the optimizer of the relational database. The job manager is responsible for coordination between the distributed executions of the data flow, as it tracks the progress and state of each operator and stream, schedules new operators, and finally monitors checkpoints and recovery. The actual data processing happens in the task manager, which is responsible for the execution of one or more operators that produce streams and report the status of them to the job manager. The task manager keeps a buffer pool to buffer the streams and the new connections to exchange the data streams among operators [19].

Other approaches

This category discusses studies that process Big Data using approaches rather than MapReduce, Spark, or Fink. There are many popular programming languages for data science as R, SAS, and SPSS. R specifically gains a lot of popularity, as it is an open-source statistical programming language with some extensions that support machine learning and data processing tasks. Currently, more than 8,000 packages could be easily downloaded from Cran. R has many limitations as, it is a single-core, single-threaded because of its sequential processing, does not support parallel processing, supports in-memory processing as data has to be loaded in ram, and finally, it is not a distributed processing. R has a package for record linkage called record linkage, which aims to facilitate the record linkage process. The record linkage package provides a way to perform and evaluate efficient record linkage methods. To reduce computation time, blocking methods could be employed [20].

There is an increasing need for interactive analysis for Big Datasets. Data scientists often use R to perform advanced analytics on the dataset, however, data analysis using R is limited by the available memory on a single machine, as it is single threading, so it is often impractical to use it on a large dataset.

Many solutions address these limitations by better I/O support, integration with Hadoop, and by designing a distributed R runtime that can be integrated with DBMS engines. Venkataraman *et al.* [21] present a SparkR, an R package with a frontend to Spark which uses Spark's distributed processing capabilities to analyze large datasets from R shell. SparkR aims to scale R programs and deploy them across some workloads.

Comparison between MapReduce, Flink and Apache Spark

Hadoop, Spark, and Flink are the top 3 Big Data technology that entered the IT market so rapidly. Spark entered the market because of the Hadoop limitations, as Hadoop does not support real-time data processing, only batch processing is supported. Besides, Hadoop MapReduce is not efficient with processing small datasets. Finally, Hadoop MapReduce has a latency time. The Apache Flink appears because of the limitations of Spark such as latency, manual optimization, and less number of algorithms. A detailed comparison is illustrated in **Table 1**.

Table 1 Comparison between Hadoop MapReduce, Apache Spark, and Apache Flink.

Criteria	Hadoop MapReduce	Apache Spark	Apache Flink
Data processing	Batch processing	Batch processing Stream processing	Batch processing Stream processing
Performance	Slower than Spark or Flink	Its stream processing is not much efficient like Apache Flink as it uses micro-batch processing.	Excellent as compared to any other data processing system
Fault tolerance	Support fault tolerance	Support fault tolerance	Support fault tolerance
Scalability	Scalable	Scalable	Scalable
Optimization	Manually optimized	Manually optimized	Has an optimizer
Latency	Higher latency than both Spark and Flink	Relatively faster than Hadoop MapReduce	Low latency and high throughput
Processing speed	Slower than Spark and Flink.	100 times faster than MapReduce (due to in memory computations of it)	Faster than Spark because of its streaming architecture
Real-time analysis	No real-time analytics	Perform real-time data analysis	Perform real-time data analysis
Machine Learning	Requires Machine Learning tool as Apache Mahout	Has its own Machine Learning library (MLlib)	Has its own Machine Learning library (FlinkML)
High availability	Configurable in high availability mode	Configurable in high availability Mode	Configurable in High Availability Mode
Duplication elimination	There is no duplication elimination	Processes every record exactly one time hence eliminates duplication	Processes every record exactly one time hence eliminates duplication

Classification of the covered studies according to the parallel processing approaches for Big Data

The techniques that handle the increasing volume of data depend on parallel processing, where the input data is partitioned into several blocks to distribute the workload between them, then these blocks will be processed using one of the technologies described above.

This paper aims to conduct a comparative study of record linkage approaches for Big Data. To accomplish our comparative study, we searched many electronic databases to determine if similar work has been performed or not and locate the most recent and relevant studies. We conducted our search using search strings constructed using combinations of keywords and synonyms. The search string included (record linkage AND big data) OR (entity resolution AND big data) OR (duplicate detection AND big data) OR (data matching AND big data) OR (deduplication AND big data) OR (MapReduce AND entity resolution) OR (MapReduce AND record linkage) OR (MapReduce AND deduplication) OR (MapReduce AND deduplication) OR (Spark AND entity resolution) OR (Spark AND record linkage) OR (Spark AND deduplication) OR (Spark AND deduplication) OR (Flink AND entity resolution) OR (Flink AND record linkage) OR (Flink AND deduplication) OR (Flink AND deduplication) OR (parallel processing approaches for Big Data) AND (Data >=2012). From the entire received list, we choose the most recent publications. We selected 21 studies from 2012 to our present date. We limited our search to only include English papers published in journals or conference proceedings. We excluded papers that are not directly related to our research and paper that have not any data about them. We also excluded news, comments, and a duplicate paper of the same study. The inclusion of the papers depends on its title and abstract where the title by itself may not sufficient to determine the inclusion of the paper as a candidate study. If the title and abstract are not enough to take the decision, then the introduction is read or even the whole paper. Papers passing the selection criteria are read in-depth to fill a pre-defined extraction form. We created an extraction form to extract relevant information from each selected paper. The extraction form is organized into 3 tables. **Table 2**: contains the relevant papers covered in the comparative study. **Table 3** contains the summary and abbreviations of the phases, properties, and techniques of the three dimensions of the comparative study, and finally, **Table 4** presents the results of the comparative study according to the three proposed dimensions.

As the topic of the research is still recent, the number of found studies is limited. The studies in this paper are divided according to the parallel processing approaches discussed before to four categories. Category 1 contains studies that adopt MapReduce based approach for resolving duplicates. This category has ten papers. Category 2 contains studies that adopt Spark-based approach for resolving duplicates. This category contains eight papers. Category 3 contains studies that adopt Flink based approach for resolving duplicates. This category has 2 papers. Category 4 contains studies that adopt other approaches rather than MapReduce, Spark, or Flink. This category also has only one paper. In the following subsections, we will explain each category.

Table 2 Studies Covered in This Research.

Study Number (SN)	Study name	Publication year
S1	Spark-based workflow for probabilistic record linkage of healthcare data	2015
S2	Distributed holistic clustering on linked data	2017
S3	Data linkage for Big Data using Hadoop MapReduce	2015
S4	Entity resolution in Big Data framework	2015
S5	Geocoding billion of address: towards a special record linkage system with Big Data	2012
S6	Big Data entity resolution from highly to somehow similar entity descriptions in the Web	2015

Study Number (SN)	Study name	Publication year
S7	Cross language duplicate record detection in Big Data	2015
S8	Elodu: Entity resolution in Big Data	2015
S9	A Machine Learning approach to census record linking	2016
S10	Minoan ER: Progressive entity resolution in the Web of Data	2016
S11	Hadoop framework for entity resolution within high-velocity streams	2016
S12	SparkER: Scaling Entity resolution in spark	2019
S13	Data matching and deduplication over Big Data using Hadoop framework	2016
S14	BigDedup: A Big Data Integration Toolkit for duplicate detection in industrial scenarios	2018
S15	Entity Resolution and data fusion an integrated approach	2019
S16	Parallel Entity Resolution with Apache Spark	2017
S17	Scalable Matching and Clustering of Entities with FAMER	2018
S18	An efficient Spark-Based adaptive windowing for entity matching	2017
S19	Parallel duplicate detection in adverse drug reaction database with spark	2018
S20	Exploring Spark-SQL-Based Entity Resolution using the persistence capability	2018
S21	Dedoop: Efficient Deduplication with Hadoop	2012

Category 1: Papers adopted MapReduce approach

MapReduce is used to parallelize the process of record linkage effectively and efficiently. Parallelizing the process of record linkage depends on effectively distribute the workload between many nodes exploiting the techniques of adaptive blocking [22]. The papers adopt this approach are the majority of the studies being explored. Ten studies adopted this approach. Next, are a summary of each study.

In S3, C *et al.* [23] proposed a method based on One Class-Clustering Tree (OCCT) to characterize the entities that should be linked together. OCCT is built using the MapReduce framework for parallel execution of one-to-many record linkage tasks to reduce their associated execution time. The objective was to match records from a dataset with their corresponding records in the other dataset using the One-Class Clustering Tree. In OCCT, each one-to-many record linkage task is executed using the Map class while the reducer is responsible for combing the results of all mapper tasks. The proposed solution of record linkage is implemented using the MapReduce framework and consisted of two steps. The first one is linkage model creation where the clustering tree is created using maximum likelihood splitting to determine which attributes will be used for building the tree. The second step is the leaf representation in which each leaf has a dataset containing matching records from the other dataset.

In S4, Kejriwal [23] discussed the record linkage in Linked Open Data (LOD). LOD is a collection of Resource Description Framework (RDF) dataset. LOD currently contains over 30 billion triples over 500 million property edges publishes in over 300 datasets. The aim of this paper is to build an unsupervised record linkage prototype, which accepts $N \geq 1$ heterogeneous datasets as inputs and outputs a set of matched entities. The current record linkage system is only designed for $N = 1$ or $N = 2$. Kejriwal [23] designed a workflow for arbitrary N, which is an area for research and involves a novel challenge. The workflow starts by accepting N heterogeneous datasets then goes through 2 MapReduce Algorithms. The first MapReduce workflow accepts N heterogeneous datasets as inputs and learns N-way Blocking schema and train N-way classifier while the second MapReduce algorithm accepts the output of the first

MapReduce algorithm and N heterogeneous datasets as inputs. The output of the second MapReduce algorithm is the clusters of matching entities.

In S5, Xu *et al.* [25] implemented a geocoding billion address using Hadoop and Amazon Web Service (AWS) cluster. They deployed it on 25 nodes of the Hadoop cluster. The system begins with a data preparation phase, which takes billions of addresses as an input to extract full addresses string, assign a unique id, and remove the redundant address. The second phase is running Intelius Address Parser IAP on Hadoop cluster setup on AWS consisting of 25 nodes. The execution time takes 38 h, which is less than any commercial tool. Although partitioning the Big Data for parallel execution is very important for efficient and effective record linkage, they did not use any blocking methods, which of course will have a great impact on execution time.

In S6, Efthymiou *et al.* [26] focused on record linkage for the web data i.e. identifying descriptions that refer to the same real-world entity. They used blocking as a pre-processing method to compare only elements in the same block. They also studied existing blocking algorithms to determine the factors that make blocking algorithms take different decisions about the potential matching of two descriptions. They proposed 3 different blocking methods. The first one is "token clusters" in which descriptions in the same block should at least share a common token. The second one is the "attribute cluster" where descriptions should at least have a common token in attributes values that have similar values overall. The third one is "prefix-infix-suffix blocking" where also the descriptions should share a common token at least in their literal values. They adopt MapReduce in evaluating each of the three blocking techniques using real data from billions of triples datasets. They perform pairwise Jaccard similarity measures between trigram sets of all attributes. They concluded that highly similar descriptions are met in central Linked Open Data (LOD) and have many common tokens in their descriptions, while somehow similar descriptions are met in the peripheral collections and have few common tokens in their attributes.

In S7, Yousef [15] discussed duplicate detection when the same object is represented in multiple languages especially the name of the object. This study presented a generic cross-language based solution architecture for duplicate record detection. The current duplicate detection tools have many limitations for detecting the object's name represented in many languages like Arabic, English, French, German, and Greek. Architecture had been proposed to support duplicate detection of the name of the object written in many languages. The proposed architecture was designed using a dictionary based on phonetic algorithms and supports different blocking techniques. The proposed duplicate record detection framework began with stating the required information about language extension, data sources, blocking options, and other parameters for the duplicate detection process. General cleaning and standardization rules are applied to the dataset containing Arabic or English languages. As the Arabic language is detected, a special treatment is applied to cover the wide range of typographical variations of Arabic. To reduce the complexity of the overall duplicate detection process, a blocking technique is performed. To detect the duplicate records, they are compared using Jaro-Winkler string similarity matching. After calculating the similarity between records, they are classified into duplicates, non-duplicates, and possible duplicates. The proposed framework used training data in order to identify the upper and lower bound of the threshold value. The final step of the framework is a quality evaluation, which is done using a confusion matrix. Implementing the above framework in Big Data is done using Hadoop and HBase.

In S8, Pham *et al.* [27] discussed and illustrated the usage of Hadoop for entity resolution in Big Data. The proposed workflow consists of four MapReduce functions. Those functions are counting frequency, token ordering, similarity join, and similarity join post-processing. Each of them is a separate java function consists of mappers and reducers. The count frequency job takes as input the dataset sorted in HDFS, finds the join fields and divide it into tokens, calculate token frequency and finally write token and their frequency as key-value pairs in a MapReduce job. The second job (token order) swaps key-value pairs and sorts the tokens. The third job (similarity join) accepts the token ordering of the two datasets as inputs. The candidate records are found using the order of tokens, and then those candidate records are checked to find records that are joined together. The final job is similarity join post-processing which filters the last job output to get clean results.

In S10, Efthymiou *et al.* [28] presented a Minoan platform for resolving similar entity descriptions. The entities are presented in the form of Resource Description Framework (RDF) DB. The proposed

platform consists of many steps that started with blocking and meta-blocking. Entities in the same block will only be compared using similarity matching algorithms. It is the only study that applies pruning methods as it accompanies blocking with meta-blocking where meta-blocking prunes or discards comparisons between descriptions that share few common blocks and are not likely to match. Minor entity resolution extended the traditional entity resolution workflow by a scheduling phase that will select the pairs of the description, which will be compared in the entity matching process and their order.

In S11, Benny *et al.* [29] proposed a Hadoop MapReduce framework for entity resolution. The proposed framework consists of six steps. In step 1, preprocessing was done where unwanted words such as stop words were removed. The output of this step was a set of entities. This output was used as an input to the second step (token blocking step). The output of this phase was a set of tokens from input data. The third phase was a blocking graph. The input of this phase was a collection of blocks to construct a blocking graph (G) from its block assignments. The nodes of the graphs were the elements. The values in the edges represented the frequency of the tokens between two elements. The next phase was the pruning phase where some edges with less weight were removed. The next step was the entity resolution, which performed the similarity computation, and then the results were passed to the mappers. Similarity computation was done using Hive user-defined function. The results of the function were passed to the reducer phase. The average of thirteen similarity measures was used to calculate the similarity between two elements. The calculated similarity was compared with a threshold value. Actually, using 13 different kinds of similarity measures is a huge number, which was not appropriate for the big dataset, as it would take a very long processing time. In the reduce phase, the average of the calculated similarity was compared with the threshold value. If it was greater than the threshold value, then the pairs were considered matched. The final step was generating rules, where the matched pairs were stored. These rules were used to test the new data and were updated periodically with the entry of new entity pairs.

In S13, Albanese *et al.* [30] proposed a method for solving entity resolution and deduplication problems using MapReduce over the Hadoop framework. The proposed method includes pre-processing, indexing by one or more key fields using slandered blocking, comparison, and classification tasks. Using more than one blocking key field to partition the dataset may generate a redundant comparison of records if two records have similarities in more than one key field. To avoid this problem, they used a method combiner MapReduce which is used as a filtering or aggregation step to prevent the map phase to write duplicate pairs (key, value) that are then processed by the reducer phase. This method operates on one or more datasets and can work on structured and semi-structured datasets. They used the standard partitional to distribute candidate records on reducers. Using a standard partitioner is vulnerable to data skew.

In S21, Kolb, *et al.* [31] devolved a tool called Dedoop (Deduplicate with Hadoop) which supports MapReduce-based Entity Resolution for large datasets. Dedoop had a GUI for its workflow that consisted of three steps; blocking, similarity computation, and matching decision of the input blocks depending on their similarity value. The final step (match classification) could depend on a trained classifier using a Machine Learning algorithm on the training dataset. Several blocking techniques could be used in the blocking, similarity computation, and match classification. Dedoop did not offer any pruning steps to reduce the number of unnecessary comparisons.

Category 2: Papers adopted Spark

Eight studies adopted this approach. Next, are a summary of each study. Study1 (S1) adopted a Spark bases to detect duplicates. In S1, Pita *et al.* [18] presented Spark-based framework of probabilistic record linkage. The proposed framework integrated three databases from different sources. They also used a bloom filter technique for anonymization before applying the linkage stage. The framework consisted of four phases:

- Phase 1: data quality assessment. Aims to identify the attributes more suitable for probabilistic record linkage. The attributes were chosen based on their co-existence in the DB, their discriminatory capabilities, and their quality in terms of missing values or null. They used SPSS for achieving these tasks.

- Phase 2: Pre-processing. Aims to apply data transformation and standardization to prepare the dataset for record linkage process such as transforming to uppercase, substitute null values with pre-defined values. For data privacy, a bloom filter has been applied to anonymize the fields before the record linkage phase takes place. Spark is used for performing the transformation and standardization tasks.
- Phase 3: Linkage process. A deterministic record linkage had been applied between two DB using direct join and probabilistic record linkage where Spark was used to perform record linkage.
- Phase 4: Evaluate data mart. SPSS was used to evaluate the produced data marts.

Study12 (S12) Gagliardelli *et al.* [32] introduce SparkER, a distributed entity resolution tool that can scale entity resolution algorithms. SparkER runs on top of Apache Spark to take off the advantage of parallel processing and computations. SparkER in its first version focuses on blocking steps and implements both structure agnostic and blast meta-blocking approaches. Then, SparkER was extended by adding entity matching and entity clustering modules. Meta-blocking aims to restrict the collection of blocking by removing the least promising comparisons, which are achieving constructing graphs where nodes represent profiles and edges, represent comparisons. SparkER consists of 3 main modules: (1) blocker: it is responsible for taking the input profiles, performs a blocking phase, and produces the candidate pairs as output; (2) entity matcher: it takes the candidate pairs generated by the blocker and label them as matched or unmatched; (3) entity cluster: takes the matcher pairs to group the similar entities on the same cluster.

Study14 (S14) Gagliardelli *et al.* [33] present dedup, a toolkit. An efficient toolkit to detect duplicate records on big data sources using Apache Spark. It can process structured and unstructured data in an efficient manner. They implement two blocking techniques to reduce the number of comparisons. The first one is schema-agnostic blocking which is employed to avoid schema alignment (it does not consider the schema) which is a heavy task in the case of different data sources. For example, the token blocking is used to generate a blocking key that will result in a high recall but a very low precision because it retains a high number of unnecessary comparisons. To overcome such a problem and improve the precision, Simonini *et al.* [34] present a schema-aware version of token blocking called loose schema-aware blocking (LSB). LSB generates clusters of similar attributes by applying LSH on their values and give them weight, then apply token blocking in which records have the same blocking key are clusters together only if the blocking key belongs to the same cluster of attributes. However, using blocking is not adequate to reduce the number of comparisons; more sophisticated techniques are required. Papadakis *et al.* [35] proposed meta-blocking which aims to solve that problem. Meta-blocking depends on creating clusters that contain entities that share the same tokens, then, a graph is constructed where nodes are entities and edges represent that they have a common token indicated by weights such that higher values of them represent a higher value of tokens. All edges have a weight less than a threshold value (the average value of all edges) are pruned. They used 3 real-world datasets to test BigDedup. They measure the performance in terms of precision, recall, and execution time. All three datasets have a ground-truth that provides the true duplicate which made it easy to test the proposed tool. BigDedup is not tested using Big Data in a distributed manner because all the three datasets have a very small size that doesn't require a cluster, so, they only perform tests on a single machine.

Study15 (S15) Beneventano *et al.* [36] presented SparkER-MOMISDF (Spark Entity Resolution Mediator envirOnment for Multiple Information Sources Data Fusion) where SparkER presented in [32] will be extended with post-processing methods to obtain one-to-one matching by integrating SparkER with the MOMIS framework. The output of SparkER will be used as an input to the MOMISDF system. MOMISDF is responsible for performing data fusion and it is extended with a method for the evaluations of data fusion results. The complete SparkER-MOMISDF workflow consists of 2 main parts: (1) SparkER: it consists of two main modules; (a) Blocker: which performs blocking of the input records and provides candidate pairs as output. (b) Entity matcher: it takes the candidate pairs provided by blocker and labels them as matched or un-matched by comparing pairs' similarity with threshold value then producing a match table of similar records. (2) the one-to-one matching module takes match table as an input (Match table may contain many to many matching) and returns a one-to-one matching. (3)

MOMISDF: it is responsible for performing data fusion of the input records using a one-to-one matching table.

They implement three strategies to perform one-to-one matching: (1) symmetric best match (Max-both): where for every record, only the best matching record from other sources is accepted. (2) Max. Weight Matching(MWM): it is the matching that has the max. A weight that maximizes the sum of the overall similarities between records in the final linkage results (3) Stable marriage (SM): a matching is stable if there are two records of different classes who both have higher similarity to each other than to their current matching model.

Study16 (S16) Chen *et al.* [37] proposed and implement a parallel and scalable entity resolution algorithm, which can run on a big data analytics platform of Apache Spark. They adopt TF-IDS weights of tokens to improve the precision and efficiency of the algorithm. The proposed algorithm is scalable for processing large datasets. It consists of four phases: data loading and pre-processing, TF-IDF computing, Similarity measure, and evaluation of the algorithm. Different levels of parallelization are used to analyze the running time of the algorithm. Spark was running on the Yarn cluster of Cloudera. They developed four servers in the cluster; one of them is a driver node and the others are working nodes. The effectiveness of the algorithm is evaluated using a golden standard dataset.

Study18 (S18) Mestre *et al.* [38] investigated the use of Spark to perform parallel entity matching using a variation of sortied neighborhood method (SNM) that uses a variety (adaptive) window size. They propose Spark Duplicate Count Strategy (SDCS ++), a spark approach for adaptive SNM that aims to enhance the performance of it. The evaluation of the approach shows an increase in the performance of parallel DCS regarding the entity matching execution time. They Proposed an approach that combines DCS ++ and its efficient parallelization using Spark, considering load balancing techniques. By using multiple spark iterations and customize data replication to allow resizing of the adaptive window. The approach also solves the data skew by applying an automatic data partitioning strategy that provides load balancing across all working nodes. They evaluate the S-DCS ++ and show that the first approach provides a better performance comparing the others by reducing the execution time. Evaluation is implemented using real data in a real cluster environment. They implemented three transformation steps to perform adaptive windowing entity matching. They improve load balancing by equally distribute the entities between workers. By doing this, each worker will perform a maximum number of window's slides.

Study 19 (S19) Wang *et al.* [39] proposed a scalable duplicate detection workflow developed using apache Spark to address the problem of duplicates in Adverse Drug Reaction (ADR) report databases. These reports often come from a variety of sources. They proposed a method that uses a KNN (K-nearest neighborhood) classifier to identify labeled report pairs to reduce the high computational cost of KNN. They partitioned the labeled data into clusters for parallel computing. The duplicate detection workflow of ADR contains the following major components: (1) report Database: it stores reports collected by regulation and new reports are continuously added to it. (2) text-processing module: it contains text processing components that cleanup text data in the report using NLP to extract useful information from it. (3) Pair-wise distance computing module: it computes the pair-wise distance among the set of reports from the reports database. (4) training databases (Labeled datasets): the system contains two temporary databases, the first one has duplicate report pairs and the second one contains a sample of non-duplicate pairs. initial duplicate/non-duplicate is done manually by a domain expert. (5) classification module: report pairs are input to a classification module that computes the scores of each pair and generates a list of duplicate pairs using a threshold value. They used KNN classifier because its results are easy to explain by human intuition. They used an exact match for numeric and categorical fields. For string fields, they used Jaccard similarity as a text similarity measure.

Study 20 (S20) Chen *et al.* [40] This paper investigates the use of Spark-SQL for efficient parallel entity resolution and explores the persistence capabilities of spark to see its effect on performance. Therefore, they optimized the baseline workflow by applying different persistence options then they evaluated the efficiency and speed. They implemented a baseline workflow with Spark-SQL considering suitable and scalable algorithms. They concluded that the runtime of the baseline Spark-SQL entity

resolution is optimized up to three times when employing the best persistence options. In addition, they also implement the same entity resolution process using spark-RDD-based to compare it with Spark-SQL-based. They found that spark-SQL has achieved an average of 2.2 times the efficiency of Spark-RDD-based entity resolution. Spark-SQL-based entity resolution process begins with loading input data into Spark dataset with a balanced distribution to all available nodes. The pre-processing step is performed to clean and standardize the input data then the blocking is performed. The blocking step includes blocking key generation and join sub-steps. They used a standard blocking method for blocking. They used a double Metaphone algorithm to transform attributes values where letters that share a similar pronunciation are transformed to the same representation to handle common typo mistakes. Next, the similarity on each attribute is calculated for candidate pairs and then aggregated into a total similarity score. The chosen algorithm of calculating similarity score is chosen according to attribute properties (i.e. Jaro-Winkler distance is suitable for string attributes). A threshold value is used to judge whether a pair is matched or not, Then the matching result is saved, and the confusion matrix is calculated with the help of an external ground truth file. No load-balancing strategy is performed to avoid data skew and no pruning techniques are used to exclude unnecessary comparisons.

Category 3: Papers adopted Flink

Two studies adopted this technique in selected studies. In S2, Nentwig *et al.* [41] proposed a distributed holistic approach that links many data sources based on the clustering of similar entities. A cluster linked data approach had been proposed to cluster entities from multiple data sources. Entities that represented the same real-world entities were placed in one cluster. The approach depended on owl: same as links and can deal with entities of different semantic. Apache Flink had been used to support parallel execution of complex tasks of big datasets as similarity computations. Blocking techniques were implemented to reduce unnecessary comparisons. The approach began by reading the input entities into a graph-processing library of Flink (Gelly Graph) (G) with vertices (V) and edges (E). The second step was applying a set of transformations to generate entity clusters (C). The approach had three phases, the first one was the initial clustering, the second one was the cluster decomposition, and the third and the last one was the cluster merge. The first phase started with preprocessing where similarity was computed for a given input edge based on vertex property values. Then, initial clustering was created where a cluster ID was assigned to each vertex. The second phase was a cluster decomposition. It consisted of many sub-phases: the first one was the type-based grouping. It aimed to split clusters into sub-components depending on the semantic type. The second sub-phase was similarity-based refinement where clusters were decomposed again by removing non-similar entities from their clusters. The last sub-phase was the creation of a unified cluster representation for each cluster based on the contained entities. The last phase was cluster merge where highly similar clusters were combined into larger ones. To avoid the complexity of comparing all clusters, a standard blocking strategy had been applied.

Study 17 (S17) Saeedi *et al.* in [42] proposed a scalable and distributed entity resolution framework for big data called FAMER(FAst Multi-source Entity Resolution system) developed used Apache Flink and a new extension for graph analytics called Gradoop. FAMER can match entities from multi-sources. It includes multiple clustering of a schema to group matching entities. It also includes new approaches tailored to multi-source entity resolution. In addition, they perform a detailed comparative evaluation of eight clustering schemas using different, real, and synthetically generated datasets.

FAMER has 2 main parts, which are the similarity graph generator and entity clustering. Similarity graph generator has three steps (blocking, pair-wise comparison, and matching classifier) which can be customized according to configuration input. In the blocking step, only entities that reside in the same block will be compared. A blocking key may cause skewed blocking size, which can result in significant runtime problems in distributed implementation. To achieve load balancing, FAMER supports the blocking-split method proposed where large blocks are separated again to be processed in several processing nodes. After blocking, all entities of the same block are pairwise compared. Currently, FAMER will support both supervised and unsupervised matching classification models. The output of the previous step is a set of matching entity pairs linked together with similarity values and stored in a similarity graph where entities are represented as vertices and the calculated similarity is represented as

edges. They integrated entities from different sources and assumed that entities in the same source are free of duplicates. FAMER currently includes eight clustering techniques that are implemented and evaluated.

Category 4: Papers adopted other approaches

This Category contains studies that adopt other approaches rather than MapReduce, Spark, or Flink. Many approaches have been used such as R and Spark. R is a language for statistical computing. It has a complete package for record linkage. In S9, Feigenbaum *et al.* [43] proposed a standard census matching technique for constructing linked samples that could be replicated across a variety of cases. The proposed technique applied a machine learning classification and text comparison in record linkage for historical data. The technique adopted a supervised learning approach. The approach began with some aspects of data preparation as choosing variables that will be used to compare records then, matching 2 censuses databases, and extracting a sample of possible matching records. The next step is building a training dataset that will be used to tune the proposed matching algorithm. The proposed approach displays how many records needed to be manually coded as matched or non-matched before tuning the algorithm. In this study, a comparison between classification models as random forest, support vector machine, and probit model had been accomplished. The last one achieved the best performance. The proposed method used well-known tools like R and Stata.

Table 3 Abbreviations Table.

Record linkage phases	Properties of dataset		Parallel processing approaches for Big Data
	Type	Size	
Data Exploration (DE)	Structured (S)	Big Data (BD)	MapReduce (MR)
Data Preparation (DP)	Simi-Structured (SS)	Very Large- Scale	Spark-based (Spk)
Blocking Techniques (BLKT)	Unstructured (US)	Dataset (VLSD)	Flink (FLK)
Similarity Methods (SM)		Small Data (SD)	Others (Stata, R, Python,…)
• Character Distance Similarity-Based (CDS)			Pruning Techniques (PT)
• Token Distance Similarity-Based (TDSB)			
• Phonetic Distance Similarity-Based (PDS)			
• Numeric Distance Similarity-Based (NDS)			
• Hybrid Approach (HA)			
Record linkage Approach (RLA)			
• Probabilistic (P)			
• Graph-Based (G)			
• Machine Learning (ML)			
• Distance-Based approaches (D)			
• Rule-Based (RB)			
• Hybrid Approach (HA)			
Quality Matrix (QM)			

Results of the comparative Study on the selected papers.

In **Table 3**, we summarize and abbreviate the phases, properties, and techniques of the three dimensions that formulate the comparative study discussed in section 2. The first one is the proposed record linkage phases, the second one is the dataset's properties (i.e. its type and size), and the third and last one is the used record linkage approach for Big Data discussed in section 3.

For size limitation, we create abbreviations of the components of three dimensions that formulate the comparative study. **Table 4** presents the results of the comparative study according to the three proposed dimensions, which can be summarized as follows:

Table 4 Comparative Analysis.

SN	Record linkage phases						Dataset (DS)		Parallel processing approaches for Big Data				
	1.DE	2.DP	3.BLKT	4.SM	5.RLA	6.QM	Type	Size	MR	SPK	FLK	Others	PT
S1	√	√	√	(CDS): Dice Distance	HA: (P) (D)	√ (Execution Time)	(S)	VLSD	X	√	X	X	X
S2	X	√	√	(CDS)	(G)	√	(SS)	BD	X	X	√	X	X
S3	X	X	X	NA	(G)	NA	(S) (SS)	BD	√	X	X	X	X
S4	X	X	√	(CDS): Cosine similarity	(G)	NA	(SS)	BD	√	X	X	X	X
S5	X	√	X	(CDS)	(D)	√	(S)	BD	√	X	X	X	X
S6	X	X	√	(CDS)	(D)	√	(SS)	BD	√	X	X	X	X
S7	X	√	√	HA: (PDS), (CDS)	(ML)	√	(S)	BD	√	X	X	X	X
S8	X	√	X	(CDS)	(RB)	√	(SS)	BD	√	X	X	X	X
S9	√	√	√	(CDS)	HA: (ML) (RB)	√	(SS)	VLSD	X	X	X	√ (using R)	X
S10	X	X	√	NA	(G)	√	(SS)	VLSD	√	X	X	X	√
S11	X	√	√	HA: (CDS), (TDS), (PDS)	(RB)	√	(SS)	BD	√	X	X	X	√
S12	X	√	√	(CDS)	(G)	√	(S) (SS)	NA	X	√	X	X	√
S13	X	√	√	(CDS)	(D)	NA	(S) (SS)	BD	√	X	X	X	√
S14	X	X	√	(CDS)	(G)	√	(S) (SS)	SD	X	√	X	X	√
S15	X	√	X	(CDS)	(D)	√	(SS)	SD	X	√	X	X	√
S16	X	√	X	(CDS)	(D)	√	(SS)	SD	X	√	X	X	X

S17	X	X	√	(CDS)	(G)	√		(SS)	VLSD	X	X	√	X	X
S18	X	X	√	(CDS)	(D)	√		(S)	VLSD	X	√	X	X	√
S19	X	√	X	(CDS)	HA: (ML) (D)	√		(S)	VLSD	X	√	X	X	√
S20	X	√	√	(CDS)	(D)	√		(S) (SS)	VLSD	X	√	X	X	X
S21	X	X	√	(CDS)	(ML)	√	(Execution Time)	(S) (SS)	VLSD	√	X	X	X	X

1. In record linkage phases section.

- Despite the importance of data explorations, it is almost a non-existing phase in the studies covered in this paper. It is only performed in only (10 % of the studies). Data exploration is very important in bringing important aspects of the datasets being examined such as outliers, and null values to take actions regarding them. Actions may include filling null values with the values of similar attributes, taking the average value of similar attributes or in some cases discard the whole record. Not paying attention to this aspect may result in mismatching some records or decreasing the accuracy of Record Linkage classifiers.
- Although Data preparation is highly affecting the accuracy of the results of record linkage classifiers, it is found only found in 62 % of the covered studies. Data preparation aims to manipulate the data to be ready for examination by Record Linkage algorithms because taking the data as it is will not guarantee to have accurate results of record linkage classifiers. Data preparation includes normalization, transformation, and standardization of the data. As an example, the male value of gender attribute may be stored as 'M' in one dataset and '1' in another dataset. If we do not transform one value to the other, we will not be able to match them.
- The blocking techniques are used in 67 % of the studies despite its importance in reducing the dimensional search space and hence affect the performance of record linkage approaches by decreasing the runtime.
- The similarity measures are not available (NA) in two studies.
- As depicted in **Figure 3**, there are many record linkage approaches used. Graph-based approaches are used in 33 % of the studies. Distance-based approaches are used in 33 % of the studies. Rule-based approaches are used in 10 % of the studies. Hybrid approaches are used in 14 % of the studies. Finally, Machine-learning approaches are used in 10 % of the studies.

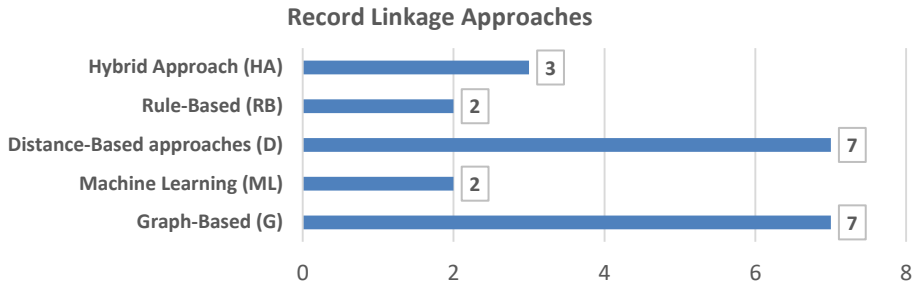

Figure 3 Record linkage approaches applied in the selected studies.

- Quality measures are used in 86 % of the studies, and not available in 14 % of them. Maybe the reasons for that are due to the unavailability of big golden datasets to test the record linkage approaches. Additionally, it is infeasible to have human experts dedicated to reviewing the accuracy of the results as it a very exhausting task and time-consuming. A summary of the results of record linkage phases is depicted in **Figure 4**.

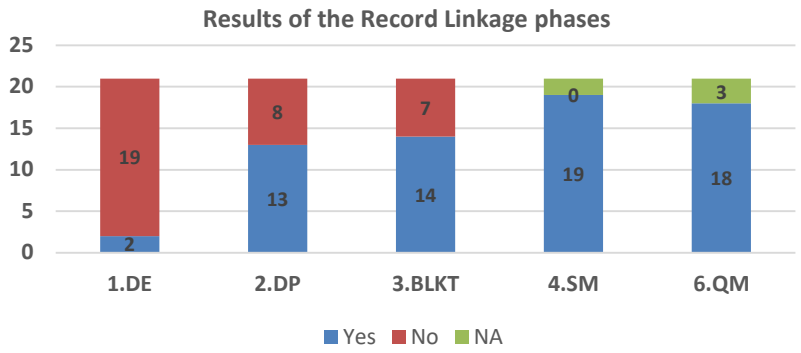

Figure 4 Summary of results of the record linkage phases in the selected studies.

2. In the dataset section.

- About (48 % of the studies) conduct their experiments on semi-structured datasets. The rest are structured datasets (28 % of the studies) and 5 studies (24 % of the studies) integrate structured and semi-structured sources as depicted in **Figure 5.**
- About (43 %) of the Record Linkage approaches are tested on real clusters using Big Datasets, whereas, the rest is examined using a Very Large-Scale Dataset (38 %), small datasets (14 %), and NA (5 %). Record linkage approaches for Big Data are never explored using un-structured dataset in the covered studies.

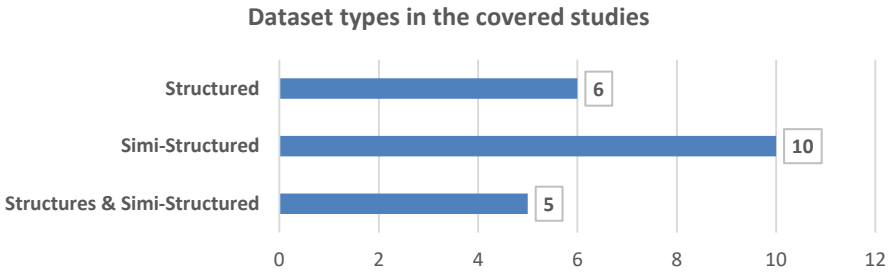

Figure 5 Summary of the dataset types in the selected studies.

3. In parallel processing approaches for Big Data section.

- Most of the studies process big data in parallel using MapReduce (48 %). Because of the limitations of MapReduce, new approaches have been adopted as Spark, Flink, and R. About 38% of the studies implement record linkage in parallel employing Spark, and 10% of them utilizing Flink, and only one study (4 %) adopted R. **Figure 6** summarizes the parallel processing approaches for Big Data that are implemented in the selected studies.

- Despite the importance of pruning techniques in reducing the number of unnecessary comparisons to gain, and therefore gaining a more effective Record linkage results. It is only applied in eight studies (38 %).

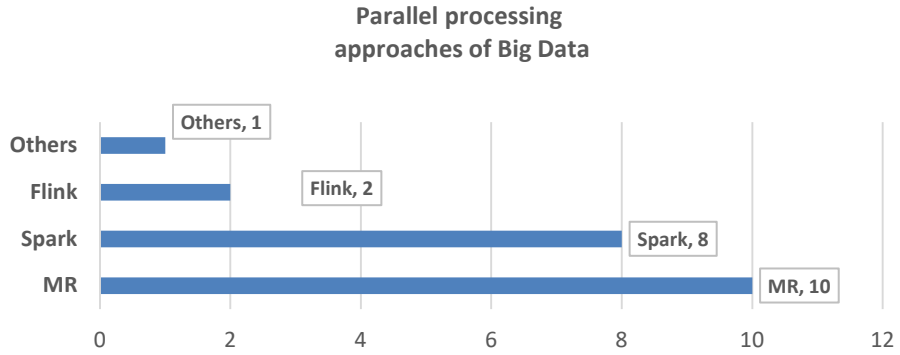

Figure 6 Summary of the used parallel processing approaches in the selected studies.

Conclusion and future work

In this paper, we present a comparative study of the available literature about record linkage for Big Data. The comparative study has three dimensions; the first one is the six phases of record linkage process; the second one is the used dataset characteristics; the third and the last one is the parallel processing approaches of Big Data. From the comparative study, we conclude, first: data exploration is almost a non-existing phase although its importance of exploring dataset being examined. Second, techniques that handle data preparation and standardization are not exclusively discussed in the literature although they can directly affect the results' quality because of the concept of garbage in garbage out. Third, handling unstructured data is never explored in the selected studies. Fourth, about half of the analyzed literature in the comparative study adopts MapReduce to handle the parallel processing of Big Data despite its limitations, therefore, there is a clear need for more researches adopted Apache Spark and Flink approaches for detecting and resolving duplicates in Big Data. Using Spark or Flink instead of MapReduce is more efficient as they guarantee high performance, low latency, and high processing speed compared to MapReduce due to their in-memory clustering computation and stream processing, therefore, Spark is the most appropriate solution in case of the need for real-time record linkage. The comparative study includes many recent researches that address the record linkage for Big Data using Apache Spark but still, more work is needed as it is still brand new. In addition, Apache Flink has rarely applied to solve the problem of record linkage for Big Data. Fifth, Pruning techniques are used in less than 40 % of the selected studies despite their effect on reducing the search space therefore, more attention should be directed to pruning techniques especially with Big Data as it will have a very great effect on the performance as blocking is not the only way to reduce the number of comparisons. As future work, we will adopt in-memory processing techniques using Spark to handle the record linkage problem for the semi-structured big dataset. We will also exploit the dimensions of the comparative study, especially the first and last one by taking it as a framework to our proposed record linkage approach for big data and apply all the steps discussed in the first dimension.

References

[1] J Manyika, M Chui, B Brown, J Bughin, R Dobbs, C Roxburgh and AH Byers. Big data: The next frontier for innovation, competition, and productivity. Available at: http://www.mckinsey.com/insights/mgi/research/technology_and_innovation/big_data_the_next_frontier_for_innovation, accessed July 2012.
[2] E Dumbill. What is Big Data? Available at: http://radar.oreilly.com/2012/01/what-is-big-data.html, accessed July 2012.
[3] TP Gondaliya, HD Joshi and H Joshi. New big things in era of digital data: Big data & big data challenges with its solution using different tools. In: Proceedings of the 10th International CALIBER, Shimla, Himachal Pradesh, India, 2015, p. 496-507.
[4] AJ Loughrey and P Deepak. *Semi-supervised and Unsupervised Approaches to Record Pairs Classification*. Multi-source Data Linkage, Springer, Cham, 2019, 55-78.
[5] F Castanedo. *Data Preparation in the Big Data Era*. O'Reilly Media, USA 1005 Gravenstein Highway North, Sebastopol, CA, 2015.
[6] XL Dong and D Srivastava. Big Data integration. *In*: Proceedings of the 2013 IEEE 29th International Conference on Data Engineering, Brisbane, QLD, Australia, 2013.
[7] H Köpcke and E Rahm. Frameworks for entity matching: A comparison. *Data Knowl. Eng*. 2010; **69**, 197-210.
[8] R Mothukuri, M Nagaraju and D Chilukuri. Similarity measures for text classification. *Int. J. Emerg. Trends Tech. Comput. Sci*. 2016; **5**, 16-24.
[9] RMA El-Ghafar, MH Gheith, AH El-Bastawissy and ES Nasr. Record linkage approaches in big data: A state of. *In*: Proceedings of the 13th International Computer Engineering Conference, Cairo, Egypt, 2017.
[10] WW Cohen, P Ravikumar and SE Fienberg. A comparison of string distance metrics for name-matching tasks. *In*: Proceedings of the International Joint Conference on Artificial Intelligence. American Association for Artificial Intelligence, 2003, p. 73-8.
[11] J Mielke. A phonetically-based phonetic similarity metric. *In*: Proceedings of the 40th Meeting of the North East Linguistic Society, MIT, 2009.
[12] S Schumacher. *Probabilistic Versus Deterministic Data Matching: Making an Accurate Decision*. Information Management Special Reports, 2007.
[13] MG Elfeky, VS Verykios, AK Elmagarrnid, TM Ghanem and AR Huwait. *Record Linkage: A Machine Learning Approach, A Toolbox, and A Digital Government Web Service*. Technical Report CSD-TR, 2003.
[14] J Domingo-Ferrer and V Torra. Distance-based and probabilistic record linkage for re-identification of records with categorical variables. *Butlletí de l'ACIA* 2002; **2002**, 27.
[15] A H Yousef. *Cross Language Duplicate Record Detection in Big Data*. *In*: Big Data in Complex Systems, Springer, Cham, 2015.
[16] J Dean and S Ghemawat. MapReduce: Simplified data processing on large clusters. *In*: Proceeding of the 6th conference on Symposium on Operating Systems Design & Implementation, Berkeley, CA, USA. 2004.
[17] AG Shoro and TR Soomro. *Big Data Analysis: Apache Spark Perspective*. *Global J. Comput. Sci. Tech. C Software Data Eng.* 2015; **15**, 6-14.
[18] R Pita, C Pinto, P Melo, M Silva, M Barreto and D Rasella. A Spark-based workflow for probabilistic record linkage of healthcare data. *In*: Workshop on Algorithms and Systems for MapReduce and Beyond, Brussels, 2015.
[19] P Carbone, S Ewen, S Haridi, A Katsifodimos, V Markl and K Tzoumas. Apache flink: Stream and batch processing in a single engine. *IEEE Data Eng. Bull.* 2015; **36**, 28-38.
[20] M Sariyar and A Borg. The RecordLinkage Package: Detecting Errors in Data. *The R J*. 2010; **2**, 61-7.

[21] S Venkataraman, Z Yang, D Liu, E Liang, H Falaki, X Meng, R Xin, A Ghodsi, M Franklin, I Stoica and M Zaharia. SparkR: Scaling R programs with spark. *In*: Proceedings of the ACM SIGMOD International Conference on Management of Data, 2016.
[22] XL Dong and D Srivastava. *Big Data Integration*. Morgan & Claypool Publishers, 2015, p. 1-198.
[23] C Bargavi and MBP Blessa. Data linkage for big data using Hadoop MapReduce. *Int. J. Comput. Sci. Technol.* 2015; **6**, 93-5.
[24] M Kejriwal. Entity resolution in a big data framework. *In:* Proceedings of the 29th AAAI Conference on Artificial Intelligence, Austin, Texas, 2015.
[25] S Xu, S Flexner and V Carvalho. *Geocoding Billions of Addresses: Towards a Spatial Record Linkage System with Big Data*. GIScience in the Big Data Age, Columbus, Ohio, 2012.
[26] V Efthymiou, K Stefanidis and V Christophides. *Big Data Entity Resolution: From Highly to Somehow Similar Entity Descriptions in the Web*. IEEE Big Data, 2015.
[27] DM Pham and TLX Vu. 2015, ELODU: Entity Resolution in Big Data. Ph. D. Dissertation, Worcester Polytechnic Institute.
[28] V Efthymiou, K Stefanidis and V Christophides. Minoan ER: Progressive Entity Resolution in the Web of Data. *In*: Proceedings of the 19th International Conference on Extending Database Technology. Bordeaux, France, 2016.
[29] SP Benny, SD Vasavi and P Anupriya. Hadoop framework for entity resolution within high velocity streams. *In*: Proceedings of the International Conference on Computational Modeling and Security, 2016.
[30] PA Albanese and JM Ale. *Data Matching and Deduplication over Big Data using Hadoop Framework*. El Servicio de Difusión de la Creación Intelectual, 2016.
[31] L Kolb, A Thor and E Rahm. Dedoop: Efficient Deduplication with Hadoop. *Proceed. VLDB Endow.* 2012; **5**, 1878-881.
[32] L Gagliardelli, G Simonini, D Beneventano and S Bergamaschi. SparkER: Scaling Entity Resolution in Spark. *In*: Proceedings of the 22nd International Conference on Extending Database Technology, Lisbon, Portugal, 2019.
[33] L Gagliardelli, S Zhu, G Simonini and S Bergamaschi. Bigdedup: A Big Data integration toolkit for duplicate detection in industrial scenarios. *In*: Proceedings of the 2018 on Transdisciplinary Engineering, Modena, 2018.
[34] G Simonini and S Bergamaschi. BLAST: A Loosely Schema-aware Meta-blocking approach. *Proceed. VLDB Endow.* 2016; **9**, 1173-84.
[35] G Papadakis, G Koutrika, T Palpanas and W Nejdl. Meta Blocking: Taking entity resolution to the next level. *IEEE Trans. Knowl. Data Eng*. 2014; **26**, 1946-60.
[36] D Beneventano, S Bergamaschi, L Gagliardelli and G Simonini. Entity resolution and data fusion: An integrated approach. *In*: Proceedings of the Symposium on Advanced Database Systems, Italy, 2019.
[37] MG Chen and HJ Sui. Parallel entity resolution with Apache spark. *In:* Proceedings of the International Conference on Electronic, Control, Automation and Mechanical Engineering, 2018.
[38] DG Mestre, CES Pires, DC Nascimento, AR Queiroz, VB Santos and TB Araujoa. An efficient Spark-based adaptive windowing for entity matching. *J. Syst. Software* 2017, **128**, 1-10.
[39] C Wang and S Karimi. Parallel duplicate detection in adverse drug reaction databases with Spark. *In*: Proceedings of the19th International Conference on Extending Database Technology, Bordeaux, France, 2016.
[40] X Chen, R Zoun, E Schallehn, S Mantha, K Rapuru and G Saake. Exploring Spark-SQL-based entity resolution using the persistence capability. *In*: Proceedings of the International Conference: Beyond Databases, Architectures and Structures. Cham, 2018.
[41] M Nentwig, A Groß, M Moller and E Rahm. Distributed holistic clustering on linked data. *In:* Proceedings of the 2017 Move to Meaningful Internet Systems. Springer, Cham, 2017.
[42] A Saeedi, M Nentwig, E Peukert and E Rahm. Scalable matching and clustering of entities with FAMER. *Complex Syst. Inform. Model. Quart.* 2018, **16**. 61-83.
[43] JJ Feigenbaum. *A Machine Learning Approach to Census Record Linking*. Working Paper, 2016.