Towards Efficient and Effective Entity Resolution for High-Volume and Variable Data

DISSERTATION

zur Erlangung des akademischen Grades

Doktoringenieurin (Dr.-Ing.)

angenommen durch die Fakultät für Informatik
der Otto-von-Guericke-Universität Magdeburg

von M.Sc. Xiao Chen

geb. am 01.08.1988 in Shandong, China

Gutachterinnen/Gutachter

Prof. Dr. Gunter Saake
Prof. Dr. Andreas Nürnberger
Prof. Dr. Andreas Thor

Magdeburg, den 02.11.2020

Chen, Xiao:
Towards Efficient and Effective Entity Resolution for High-Volume and Variable Data
Dissertation, University of Magdeburg, 2020.

Abstract

Entity Resolution (ER), as a process to identify records that refer to the same real-world entity, faces challenges that big data has brought to it. On the one hand, high-volume data forces ER to use blocking and parallel computation to improve efficiency and scalability. In this scenario, we identify three limitations: First, facing abundant research on parallel ER, a thorough survey to overview the current state and expose research gaps is missing. Second, efficiency impacts by choosing different implementation options from big data processing frameworks are unknown. Last, an in-depth analysis and comparison of the state-of-the-art block-splitting-based load balancing strategies are not provided. Therefore, correspondingly, we first conducted a systematic literature review on parallel ER and report our findings. Then we explore three Spark implementations of two scenarios of a conventional ER process and expose their respective efficiency and speed-up. Last, we theoretically analyze and compare two state-of-the-art block-splitting-based load balancing strategies, propose two improved strategies, and then empirically evaluate them to conclude the important factors for a block-splitting-based load balancing strategy. On the other hand, facing variable data, we identify two shortcomings. First, confronting variable data with different types of attributes, word-embedding-based similarity calculation can provide uniform solutions, but the effectiveness may be lowered for attributes without semantics. Second, facing variable data from broad domains, training data required for learning-based classification may not be available leading to expensive human labeling costs. Existing committee-based active learning approaches for ER to reduce human labeling costs cannot provide balanced and informative initial training data and compromise the accuracy of their committees to provide different classification voting results. Therefore, correspondingly, we first propose a hybrid similarity calculation approach by choosing traditional syntactic-based or word-embedding-based similarity measures based on the properties of attributes to achieve higher effectiveness. Then we propose HeALER to overcome the aforementioned drawbacks of committee-based active learning ER approaches. We empirically demonstrate the improvements of our proposed approaches on both real and synthetic datasets.

Zusammenfassung

Die Entitätsauflösung als ein Prozess zur Identifizierung von Datensätzen, die sich auf dieselbe reale Entität beziehen, steht vor Herausforderungen, die Big Data mit sich gebracht hat. Einerseits zwingt die großvolumige Data die Entitätsauflösung dazu, blockbasierte und parallele Berechnung zu verwenden, um die Effizienz und Skalierbarkeit zu verbessern. In diesem Szenario werden drei Einschränkungen festgestellt: Erstens, angesichts der umfangreichen Forschung zur paralleler Entitätsauflösung fehlt eine gründliche Umfrage, um den aktuellen Forschungsstand zu erhalten und Forschungslücken aufzudecken. Zweitens sind Auswirkungen auf die Effizienz den verschiedenen Implementierungsoptionen aus Big-Data-Verarbeitungs-Frameworks nicht bekannt. Schließlich wird der eingehende Vergleich der blockaufteilungsbasierte Lastausgleichsstrategien nicht bereitgestellt. Dementsprechend führten wir zunächst eine systematische Literaturrecherche zur parallelen Entitätsauflösung durch. Anschließend untersuchen wir drei Spark Implementierungen von zwei Szenarien eines herkömmlichen Entitätsauflösungsprozesses, um deren jeweilige Effizienz zu bewerten. Zuletzt analysieren und vergleichen wir theoretisch zwei typische blockaufteilungbasierte Lastausgleichsstrategien, schlagen zwei verbesserte Strategien vor und bewerten sie umfassend, um die wichtigen Faktoren für eine blockaufteilungbasierte Lastausgleichsstrategie zu ermitteln. Auf der anderen Seite stellen wir angesichts variabler Daten zwei Mängel fest. Erstens kann die worteinbettungsbasierte Ähnlichkeitsberechnung einheitliche Lösungen liefern, wenn vielfaltige Daten mit unterschiedlichen Attributtypen konfrontiert werden. Die Effektivität kann jedoch für Attribute ohne Semantik verringert sein. Zweitens sind angesichts vielfalitiger Daten aus weiten Bereichen möglicherweise keine Trainingsdaten verfügbar, die für den lernbasierten Klassifizierungsschritt erforderlich sind, was zu teuren Kennzeichungskosten führt. Bestehende komitee-basierte Aktiveslernensansätze für die Entitätsauflösung zur Reduzierung der Kennzeichungskosten können keine ausgewogenen und informativen Daten für die erste Trainingsdaten liefern und die Wirksamkeit ihrer Komitee für unterschiedliche Abstimmungsergebnisse der Klassifizierung kompromittieren. Dementsprechend schlagen wir daher zunächst einen hybriden Ähnlichkeitsberechnung vor, indem wir traditionelle syntaktische oder worteinbettungsbasierte Ähnlichkeitsmaße basierend auf den Eigenschaften von Attributen auswählen, um eine höhere Effektivität zu erzielen. Dann schlagen wir auf heterogenen Komitees basierenden Ansatz für aktives Lernen (HeALER) vor, um die oben genannten Nachteile zu. Wir demonstrieren empirisch die Verbesserungen unserer vorgeschlagenen Ansätze sowohl für reale als auch für synthetische Datensätze.

Acknowledgements

I guess, pursuing a Ph.D. should be one of the most challenging things in my life. It would have been impossible without the support of all the lovely people I mentioned in the following.

It has been years since I started my Ph.D. study. However, I would always be grateful to Prof. Gunter Saake and Dr. Eike Schallehn for allowing me to start this journey. Moreover, your expert advice and encouragement guide me to complete the different stages of my Ph.D. step by step.

I would like to express my gratitude to Prof. Andreas Nürnberger and Prof. Andreas Thor for taking the time to evaluate my thesis, and also the other professors Prof. Sanaz Mostaghim, Prof. Till Mossakowski, and Prof. Myra Spiliopoulou for constituting my defense committee.

I am not alone on my Ph.D. journey, due to the company of my colleges of the DBSE group. I would like to thank David Broneske, Gabriel Campero Durand, and Roman Zoun, you helped me not only in my research but also helped me out of the darkness and find the beautiful scenery along my Ph.D. journey. Many Thanks to Yang Li, Juliana Alves Pereira, Yusra Shakeel, and Sabine Wehnert, I will always remember our relaxing conversations and the support you gave me. I would like to thank Ziqiang Diao as well for helping me apply for my scholarship and start my Ph.D. study. Besides, thanks to David, Basti, and Jacob, for allowing me to join your pizzaflix group, which is the precious memory of my life.

I would like to thank the Chinese Scholarship Council and the Graduiertenförderung for financial support.

At last, endless thanks to my parents, especially for the hard time during the Corona period, without your love, encouragement, and delicious meals as well, I cannot complete my thesis. The biggest thanks to my dear husband for being the doctor of my Ph.D. life. Of course, I should also thank my two lovely sons, without you, I should have finished my Ph.D. several years earlier. :)

Contents

List of Figures

List of Tables

List of Code Listings

1. Introduction

Within a single information system or across diverse information systems, there may exist different descriptions for a given real-world entity. These differences may result from unexpected errors (such as typos, wrongly placed data), different customs (such as non-uniform formats, the use of abbreviations), unavoidable alterations (such as ages, family names for women after marriage) and other possibilities. Such errors and inconsistencies can limit the applicability of the data for analysis purposes, and, accordingly, reduce the business value of the data. This kind of problems is common and exists in diverse areas. To solve such problems, digital records that refer to the same real-world entity need to be identified. This process is called Entity Resolution (ER) [Elmagarmid et al., 2007; Fellegi and Sunter, 1969]. Research on ER in computer science dates back to the middle of the last century and has thrived so far. Various techniques and algorithms[1] have been developed. Despite their maturity, the development of big data, which is characterized by the four Vs - volume, variety, velocity, and veracity [Schroeck et al., 2012; Zikopoulos et al., 2011], has brought new challenges for the research on ER and in turn calls for new approaches as solutions. This thesis touches on the first two Vs: volume and variety.

ER on High-Volume Data: The first V: Volume means the exponentially increasing data amount. In this context, the common solution - pair-wise ER, as an intrinsically time-consuming task with a complexity of $O(n^2)$ [Christen, 2012b; Efthymiou et al., 2017], has to tackle the efficiency problem to make its runtime acceptable. Besides developing specific efficient algorithms for similarity calculation and classification, there are two directions proposed to improve ER efficiency. On the one hand, blocking is used to reduce the search space of ER, in which records are divided into blocks and only records in the same block need to be compared with each other [Christen, 2012a; Papadakis et al., 2016]. On the other hand, instead of being performed serially, the ER process can be sped up by using parallel computation [Chen et al., 2018b].

In this scenario, there are the following three limitations identified for ER when facing the data volume challenge - First, to date, research on parallel ER has already

[1] For a survey on approaches for ER, we refer to Christen [2012b].

reached a prosperous state. However, no thorough survey of parallel ER has been published, which leads to the difficulty of overviewing the state-of-the-art and recognizing research gaps. Therefore, a survey with a focus on parallel ER is required. Second, to ease the implementation of parallel ER, big data processing frameworks are commonly employed because of their simple programming model [Pavlo et al., 2009] and their applicability to the ER problem with blocking [Kolb et al., 2012a]. Although different implementation options provided by big data processing frameworks are available, most of the existing research chooses one of them randomly (mainly low-level APIs) without considering their effects on the ER process, which may not provide the highest efficiency in the context of the same ER scenario. Therefore, a comparison study is needed to show how different implementation options affect the performance of ER processes. Last, the generated blocks of ER are often skewed, because the number of records that have the same blocking key has high variability. With the default data partitioning strategies of big data processing frameworks, blocks cannot be divided and each block has to be assigned as a whole to reducers. In this case, the reducer with overpopulated blocks will become the straggler of the whole cluster and dominate the overall runtime. Therefore, specific skew handling techniques have been developed to improve the speed-up of parallel ER [Gomes Mestre and Pires, 2013; Hsueh et al., 2014; Karapiperis and Verykios, 2015; Kolb et al., 2012b; McNeill et al., 2012; Yan et al., 2013a]. Therein, block-splitting-based load balancing approaches are simple but effective by splitting the overpopulated blocks into smaller sub-blocks. For state-of-the-art approaches BlockSplit and BlockSlicer, they split the overpopulated blocks in different ways. However, the comparison between two approaches is quite limited, and only speed-up experiments with fixed settings of map and reduce partitions are represented in [Gomes Mestre and Pires, 2013]. How well they work under different circumstances and whether they have some drawbacks need to be answered to make better block-splitting-based solutions for parallel ER.

ER on Variable Data: The second V: Variety means that big data covers broader domains and has various forms. In this circumstance, choosing suitable similarity measures is more challenging for domain experts. Furthermore, due to the more complicated data types including semantics, traditional similarity measures cannot fulfill the need to accurately calculate the similarity between records. Therefore, approaches using word embedding have been proposed aiming to provide a general solution for data with different properties [Ebraheem et al., 2018; Kooli et al., 2018; Mudgal et al., 2018]. However, indiscreetly using word embedding for all types of data may not provide the optimal accuracy, and adding an extra word embedding step to the similarity calculation phase may have negative effects on efficiency. Therefore, guidelines on how to choose proper similarity measures are required.

Besides the above challenge, data variety also increases the amount and difficulty of manually labeling data to provide training data for its learning-based classification step, as the ground truth needs to be provided for each type of data. To alleviate this situation, active learning has been employed for ER to reduce the human labeling effort, by selecting the most informative data for labeling so that much less training data is required to provide comparable accuracy [Arasu et al., 2010; Bellare et al., 2013; de Freitas et al., 2010; Isele and Bizer, 2013; Sarawagi and Bhamidi-

paty, 2002; Tejada et al., 2001]. Therein, the committee-based active learning, which chooses data with the most disagreement of voting results of the committee as the most informative data, has not been sufficiently explored for ER. With existing solutions [Sarawagi and Bhamidipaty, 2002; Tejada et al., 2001], the selected initial training data is not balanced and informative enough, and their homogeneous committees have to comprise their accuracy to achieve diversity of the committee, i.e., their homogeneous classifiers are trained with partial training data or sub-optimal parameter settings. Therefore, improvements on both initial training data selection and employed committees are still required.

Based on the identified limitations, we envision our research goal of providing the most suitable ER solutions under different big data scenarios by a specialized and powerful framework and recognize the following research questions:

For the data **volume** challenge,

- *RQ 1.* What are the state-of-the-art and the latent research directions of parallel ER?
- *RQ 2.* How do different implementation options provided by big data processing frameworks affect the performance of ER processes?
- *RQ 3.* How well can the state-of-the-art block-splitting-based load balancing strategies solve the load imbalance problem for parallel ER and what can be improved?

For the data **variety** challenge,

- *RQ 4.* Based on the availability of both traditional similarity functions and the word-embedding-based similarity calculation approach, how to choose between them to provide ER the highest effectiveness?
- *RQ 5.* How to select a balanced and informative initial dataset and form an effective committee for committee-based active learning ER?

To answer these RQs, we present five major contributions, which are the stepping stones towards the envisioned goal, and structure them in this thesis as follows:

- In **Chapter 3**, we present the Systematic Literature Review (SLR) results on parallel ER, which answers *RQ 1.* Our SLR resulted in a corpus of 58 articles, from which existing work relates to parallel ER has been overviewed and classified based on three sets of extracted criteria. Furthermore, based on the conclusion and discussion of existing solutions, we found out several possible directions for future research.
- In **Chapter 4**, we present the first part of our research to deal with the data volume challenge. To answer *RQ 2*, we make a comparison study of three Spark-based implementations on two ER scenarios, which use RDD, DataFrame, and Datasets APIs respectively. We compare their performance

using various synthetic datasets in different sizes and with different duplicate percentages to guide choices to enable the most efficient solution for parallel ER.

- In **Chapter 5**, we introduce the other part of our research under the data volume challenge, which addresses *RQ 3*. We first theoretically analyze and conclude the advantages and disadvantages of the state-of-the-art block-splitting-based approaches BlockSplit and BlockSlicer, then propose two improved strategies TLS and BOS. At last, we comprehensively evaluate the four strategies with multiple datasets in different properties and conclude the important factors that affect the performance of a block-splitting-based load balancing strategy.

- In **Chapter 6**, we address the first research question *RQ 4* under the data variety challenge and propose a hybrid similarity calculation method for ER. We compare its effectiveness with purely traditional similarity measures or word-embedding-based approaches on both real and synthetic datasets. Our results show the necessity to use hybrid similarity calculation to improve accuracy when the dataset is mixed with semantic and non-semantic attributes.

- in **Chapter 7**, to answer the other research question *RQ 5* related to the data variety challenge, we propose a heterogeneous-committee-based active learning approach (HeALER) for ER, which is proved to be more effective than two other committee-based active learning approaches based on our results on two benchmark ER datasets.

Related work for each research direction is provided in their corresponding chapters. Besides, to support the five main chapters, we provide background knowledge in **Chapter 2**. We conclude this thesis and discuss future work in **Chapter 8**.

2. Background

> This chapter shares material with the Open Journal of Big Data article "Cloud-Scale Entity Resolution: Current State and Open Challenges" [Chen et al., 2018b] and the DEXA-BDMICS'18 paper "Performance Comparison of Three Spark-Based Implementations of Parallel Entity Resolution" [Chen et al., 2018a].

In this chapter, we provide background knowledge to make the thesis self-contained. Firstly, in Section 2.1 we present the required knowledge in the domain of Entity Resolution (ER), which includes the basic concepts of ER, the steps of its pair-based solution, and a brief introduction of its specialized area parallel ER. Afterward, in Section 2.2, we describe the metrics we use for our research to evaluate ER approaches' effectiveness, efficiency, and scalability. At last, we introduce the tools we use to support our research, which includes the big data processing framework Apache Spark and the personal data generator GeCo for ER in Section 2.3.

2.1 Entity Resolution Preliminaries

In this section, we introduce the preliminaries to understand ER, which starts with its definition, application, and research goals in Section 2.1.1, continues with its main solution pair-wise ER in Section 2.1.2, and ends with an introduction of parallel ER in Section 2.1.3.

2.1.1 Basic Concepts of Entity Resolution

ER is the process of identifying records that refer to the same real-world entity. It is also known under several other names. In the general field of computer science it is referred to as data matching [Christen, 2012b], record linkage [Fellegi and Sunter, 1969], de-duplication [Sarawagi and Bhamidipaty, 2002], reference reconciliation [Dong et al., 2005], or object identification [Huang and Russell, 1998]. In the database domain, ER is tightly related to the techniques of similarity joins [Kolb et al., 2011a].

Today, ER plays a vital role in diverse areas, not only in traditional applications of the census or health care but also for Internet-based applications such as social media, online shopping, web searches, business mailing lists. In some domains, like the Web of data, ER is a prerequisite to enable semantic search, interlink descriptions, and deep reasoning. It is also an indispensable step in data cleaning [Hernández and Stolfo, 1998; Sarawagi, 2000], data integration [Hernández and Stolfo, 1995], and data warehousing [Brizan and Tansel, 2006].

The use of computer techniques to perform ER dates back to the middle of the last century. Since then, researchers have developed various techniques and algorithms for ER, fueled by its applications in many fields. From its early days, there have been two general goals: efficiency and effectiveness, which means how fast and how accurately an ER task can be solved. In recent years, on the one hand, to deal with the large and often unpredictable scale of the data, the ER solutions are also required to be scalable. On the other hand, choosing suitable algorithms and providing training data or ground truth for ER requires human effort, another goal - involving less human effort should also be pursued. In conclusion, solutions should consider these four aspects - efficiency, effectiveness, scalability, and less human effort to better serve the ER problem.

To solve ER problems, pair-wise ER and collective ER are two directions. Pair-wise ER is the most commonly used, it solves ER problems through local evidence by comparing each possible record pair without considering the relationships among records [Christen, 2012a]; collective ER aims to group similar data objects into the same cluster, and solves ER problems by also using global evidence (relational information among records). Compared to pair-wise ER, collective ER may improve the effectiveness of ER results at the expense of increased computation cost when the relational information is available [Indrajit and Lise, 2006]. To provide solutions for broad ER scenarios, in this thesis, we focus on issues of pair-wise ER, which will be introduced in the next section.

2.1.2 Pair-Wise Entity Resolution

A typical pair-wise ER workflow includes five major steps - data preprocessing, blocking, pair-wise comparison, classification, and evaluation [Christen, 2012b], which is shown in Figure 2.1. In Figure 2.1, the green boxes correspond to the five steps. The main existing techniques developed for each step are concluded in the right side. In the following, the details of these five steps will be introduced.

Data preprocessing: The input data that needs to be resolved is usually noisy, inconsistent, and with low data quality. Furthermore, some following steps may require the data to be in a specific format as a precondition to work properly. Therefore, before the data is processed for ER, data preprocessing is required as the first step to clean and standardize the input data. In some cases due to specific blocking or comparison techniques, data tokenization and segmentation are also included in this step [Churches et al., 2002].

Blocking: Pair-wise ER is a time-consuming and compute-intensive task. For an input dataset with n records, the number of comparisons for an ER task is $n(n-1)/2$, which means that its computing complexity for n records is $O(n^2)$.

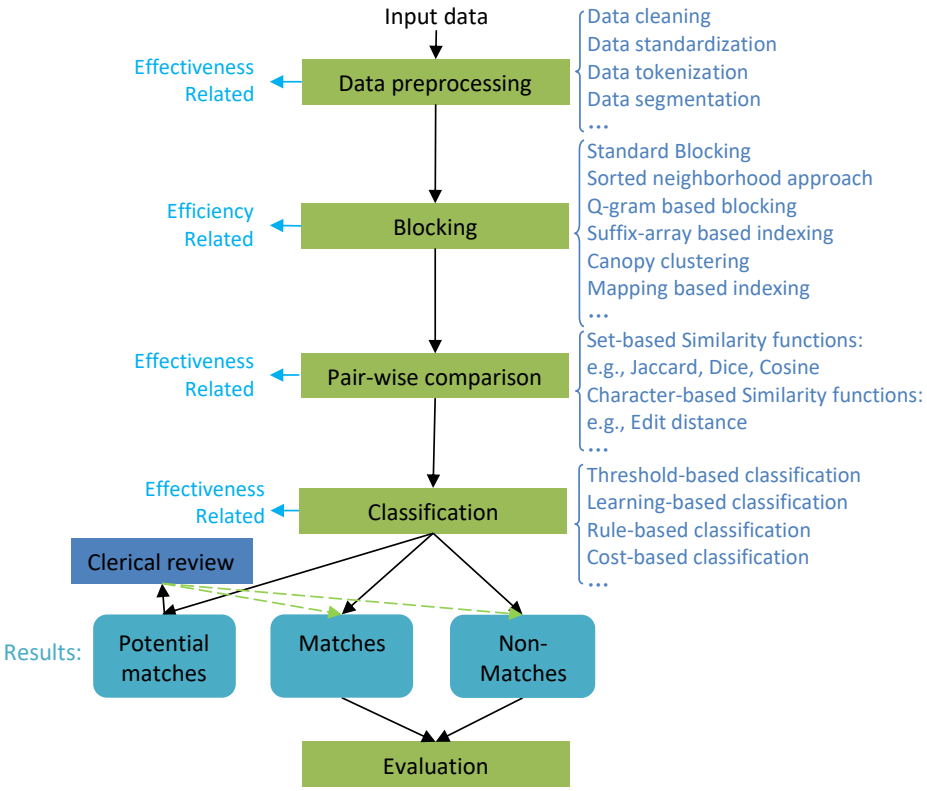


Figure 2.1: The general process of entity resolution, based on [Christen, 2012b].

As a result, the computing time for solving an ER task is dramatically extended, when the number of records increases. Therefore, blocking is often adopted as the second step to reduce the search space of ER and lower its computing complexity to $O(n)$ for fixed block size. It splits the whole input dataset into blocks and then compares only entities within the same block. The splitting criterion is called the blocking key. However, blocking may harm effectiveness due to the possibility that matching records are assigned to different blocks. Therefore, it is of great importance to keep good blocking quality, which relies on the defined blocking key and the chosen blocking strategy. Most of the blocking strategies require to define blocking keys, which are often defined by domain experts based on experience or initial data exploration steps. Both the blocking strategies and the blocking keys are crucial for a high blocking quality, and their relationship is orthogonal [Christen, 2012a].

There have been many blocking strategies developed and they can be categorized into the following six types.

- The traditional blocking [Fellegi and Sunter, 1969] is *standard blocking*, which is simply assigning all records with the same blocking key to one block.

- *Sorted neighborhood approach* [Hernández and Stolfo, 1995] firstly sorts all records based on a sorting key (similar to a blocking key), then uses a fixed-size or adaptive sliding window to split the records. Records enclosed in one sliding window belong to the same block.

Applying the above two blocking strategies can lose a large number of true matches when the blocking keys for true matches are often not identical due to the dirty and noisy input dataset. To overcome this shortcoming, the following two blocking strategies have been proposed.

- *Q-gram or n-gram based blocking* [Navarro et al., 2000] converts the original blocking key to a list of sub-strings, and then each sub-string is one independent blocking key. In this way, each record is assigned to several blocks to increase the possibility that true matches can be kept in the same block.
- *Suffix-array based indexing* [Aizawa and Oyama, 2005] is quite similar to q-gram or n-gram based blocking, it differs in the way of generating substrings by using suffixes.

Splitting the whole dataset into blocks can also be considered as a clustering problem. Canopy clustering [McCallum et al., 2000] and mapping-based blocking [Jin et al., 2003] are two clustering-based approaches for ER's blocking step.

- *Canopy clustering* can efficiently calculate the distances between the blocking keys and insert records to one or more overlapping clusters.
- *Mapping-based blocking* first maps the original blocking key into a multi-dimensional space, and then objects are extracted, lastly similar objects are inserted into the same block.

To evaluate a blocking strategy, two common metrics are reduction ratio (RR) and pairs completeness (PC) [Elfeky et al., 2002]. RR means how much percent of comparison is reduced by blocking and is calculated as follows:

$$RR = 1 - \frac{candidate\ comparisons\ after\ blocking}{entire\ comparisons\ before\ blocking} \tag{2.1}$$

PC means how much percent of true matches is remained in the candidate comparisons after blocking and is calculated as follows:

$$PC = \frac{remaining\ true\ matches\ after\ blocking}{entire\ true\ matches\ before\ blocking} \tag{2.2}$$

Generally speaking, using a relatively simple strategy (standard blocking or sorted neighborhood) is efficient and can reach a high RR but loses more true matches (a low PC), while using more complex strategies (the later-introduced four strategies) improves the PC, but it leads to a longer blocking process and a lower RR.

Another group of functions is orthogonal to the aforementioned blocking techniques, called phonetic encoding functions. They are designed to capture common transcription mistakes that people might make when recording information based on what they hear from speakers. With those functions, the letters that share a similar pronunciation are transformed to the same representation. This helps to recognize true matches and improve PC. Common encoding functions (mainly for name and address

attributes) include (fuzzy) Soundex [Holmes and McCabe, 2002; Odell and Russell, 1918; Zobel and Dart, 1996], Phonex [Lait and Randell, 1996], Phonix [Christen, 2006], NYSIIS [Borgman and Siegfried, 1992], Oxford algorithm [Gill, 1997] and Double-Metaphone [Philips, 2000].

Details of blocking strategies can be found in Christen [2012a,b].

Pair-wise comparison: The third step is the essential one of the ER process, which makes ER a compute-intensive and time-consuming task. For the pair-wise comparison, a similarity function is used to estimate how similar a pair of records is. Traditionally, a record is described with several attributes in various types. The most common type of attribute is strings. To calculate string similarities, there are the following groups of string similarity functions that can be used (we explain those similarity functions that we used in our research with details, while for others we only describe them briefly).

- One group is *distance-based* similarity, such as (Damerau -) Levenshtein distance [Damerau, 1964; Levenshtein, 1966], Affine gap distance [Waterman et al., 1976], Smith-Waterman distance [Smith et al., 1981]. They are defined based on the number of character operations required to transform one string to another [Deng et al., 2014].

- Another group is *set-based* similarity. Its input is a set of tokens of the two compared records, where a token can be either a word or an n-gram/q-gram [Navarro et al., 2000] and can be generated from data preprocessing steps. First, the number of common tokens is counted, then overlap coefficient, Jaccard coefficient [Dillon, 1983], or Dice efficient [Dice, 1945] can be used to calculate the similarities of record pairs.

- A special group is the *Jaro and Winkler* string comparison methods [Jaro, 1980], which is suitable for calculating the similarities for names. Because we use Jaro-Winkler in our research, we introduce them in the following with details. The Jaro similarity value is calculated as:

$$sim_{jaro}(s_1, s_2) = \frac{1}{3}(\frac{c}{|s_1|} + \frac{c}{|s_2|} + \frac{c-t}{c}) \tag{2.3}$$

  Therein, c is the number of common characters within half the length of the longer string, and t is the number of swapped characters in two strings.

  Jaro-Winkler similarity function increases the similarity between two strings when their initial characters are the same. It is calculated as:

$$sim_{winkler}(s_1, s_2) = sim_{jaro}(s_1, s_2) + (1.0 - sim_{jaro}(s_1, s_2))\frac{p}{10} \tag{2.4}$$

  Therein, p $(0 \leqslant p \leqslant 4)$i s the number of common characters of the prefixes of the strings.

- For *attributes with several words*, Monge-Elkan [Monge et al., 1996], extended Jaccard [Dillon, 1983], SoftTFIDF [Cohen et al., 2003] approaches have been proposed. Besides, there are also other similarity measures developed, such as longest common substring comparison, bag distance, compression distance and so on. Details can be found in Christen [2012b].

However, the quality of ER results essentially relies on the correct selection of similarity measures, which is determined by domain experts based on an elaborate analysis of record attributes and their experiences. Except for the human effort, the above-mentioned traditional methods are unaware of semantics. As a result, the accuracy may be limited for cases where words with similar semantics are expressed in different ways. Recently, word embedding is proposed for entity resolution, aiming to overcome the aforementioned limitations of traditional solutions [Ebraheem et al., 2018; Kooli et al., 2018; Mudgal et al., 2018].

- Given two attribute strings of a record pair, a *neural word embedding* model first maps each word of the attribute string from the vocabulary to vectors of real numbers [Yoshua et al., 2003], then *cosine similarity function* is used to get the final result. Word2vec [Mikolov et al., 2013a], GloVe [Pennington and R. Socher, 2014] and FastText [Bojanowski et al., 2016] are three popular neural word embedding models.

Except for strings, the numerical value is another common type for attributes. To calculate the similarity between numerical values, absolute difference, percentage difference, or euclidean distance can be used.

Classification: After receiving the similarity scores for record pairs as intermediate results, the classification step will make the decision, whether a pair of records is considered as a match or non-match, i.e. do they refer to the same real-world entity or not. There are various classification methods developed, which we will introduce in the following:

- *Simple threshold-based classification* first sums up the similarities of all attributes, and then calculates an average similarity score [Christen, 2012b]. If the average similarity score is higher than the preset threshold value, the record pair is classified as a matching pair, otherwise as a non-matching pair. It is simple to implement, however, using it by summing up the similarities of all attributes loses detailed information contained in the separated attribute similarities.

- *Probabilistic classification* sets two cutoff threshold values to divide record pairs into three groups. The higher threshold value is used to get matching pairs, whose similarity scores are higher than it. Similar to non-matching pairs, whose similarity scores are lower than the lower threshold value. Those record pairs, whose similarity scores are between the low and high threshold values, belong to the potential match group. The record pairs in the potential match group will be reviewed by human experts for the final decision. Besides, the calculation of a record pair's similarity score considers different weights for different attributes and the weights depend on both attribute characteristics and actual attribute values. The two cutoff threshold values are determined by an a priori error bound, which is hard to determine and hinders its usability [Newcombe and Kennedy, 1962; Newcombe et al., 1959].

- *Cost-based classification* analyzes the ER scenarios to determine the costs of two possible types of errors [Verykios et al., 2003]. The two types of errors are:

one true non-match record pair is wrongly resolved as a matching pair, or one true match record pair is wrongly recognized as a non-matching pair. Based on the analysis result, the classification would try to avoid the errors that cost more.

- *Rule-based classification* sets rules to classify record pairs to matches or non-matches [Hernández and Stolfo, 1995]. The rules can be set manually or learned. The learned rules classification can be considered as the learning-based classification, which will be introduced next.

- *Learning-based classification* firstly trains a classifier on a training dataset with available matching or non-matching labels, then the trained classifier is used to classify pairs with matching or non-matching status [Mitchell et al., 1997]. So far, there have been various classification techniques developed to train classifiers, we introduce some popular ones, which are also used in our research:

  - *Decision Tree (DT)* is a tree-structured and one of the most commonly used classification model. It recursively partitions the feature space by choosing the splitting criteria with the biggest information gain. It has the advantages of reasonable accuracy, inexpensive computation, and good interpretability, but may face overfitting problems [Segaran, 2008].
  - *Support Vector Machine (SVM)* constructs the maximum-margin hyperline, which has the largest distance between the two classes. Those records that are closest to the hyperline are the support vectors. It can provide accurate and fast results, once the parameters for the datasets are properly set. However, it is a black-box model and quite hard to interpret [Segaran, 2008].
  - *Logistic Regression (LR)* uses a logistic function to calculate the probability that the record belongs to one of the classes [Draisbach et al., 2012]. It is widely used because of its efficiency and high interpretability, but it only works for linear problems.
  - *Naive Bayes (NB)* algorithms assume the independence between attributes and calculate the conditional probabilities of one category of a classification problem based on Bayes theorem. It is computationally efficient, but may not be accurate due to the feature independence assumption [Rennie et al., 2003].
  - *K-Nearest Neighbor classifier (KNN)* is a special case in classification algorithms, because it does not learn any model, but stores and uses the training data directly. It considers the k closest training examples for each record to decide on the class membership of the record [Cover and Hart, 1967].
  - One-vs-Rest (OvR) [Bishop, 2006] is a strategy for multi-class classification by considering it as multiple binary classifications. A single classifier is trained using a base classifier considering one class as the positive class and all other classes as the negative class. All classifiers provide their

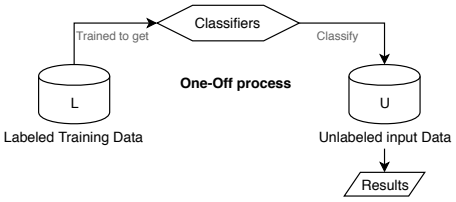


Figure 2.2: A learning-based classification process.

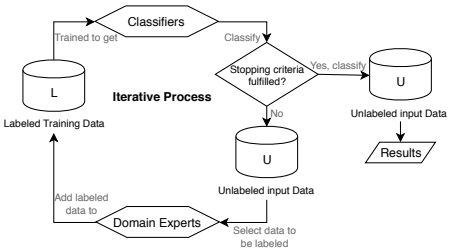


Figure 2.3: An active-learning-based classification process.

confidential value that a record belongs to its corresponding class. The output is determined by the classifier with the highest confidential value.

– *Ensemble learning* combines multiple classifiers to make decisions for classification problems. Bagging and boosting are two common methods for ensemble learning. Bagging trains multiple independent classifiers, each is trained by sampling with replacement. *Random Forest (RF)* is a bagging algorithm and is an ensemble of decision trees for achieving higher accuracy and reducing overfitting risks.

  Boosting is an iterative approach to generate a strong classifier by training an ensemble of weak classifiers and reducing the training error by selecting a subset of training data for the next iteration, which is misclassified in the previous iterations [Zhang and Ma, 2012]. *Gradient-Boosted Tree (GBT)* is a boosting algorithm, which iteratively trains decision trees by reducing a loss function. *XGBoost* is an open-source package, implements *GBT* with additive optimizations, such as a sparsity aware algorithm for handling sparse data and a theoretically justified weighted quantile sketch for approximate learning [Chen and Guestrinand, 2016].

However, training the classifiers requires a large amount of training data to achieve satisfactory accuracy, which has to be labeled by humans. As a means to reduce human effort, active learning (AL), which is a specific branch of machine learning (ML), is proposed to deal with this problem and will be introduced next.

- *AL based classification*: Compared to a normal one-off ML process (see Figure 2.2), an AL process (see Figure 2.3) is interactive and iterative. It reduces the number of required training data to achieve the desired accuracy by querying experts to label only the most informative data for each iteration and adding these into the training data. Then classifiers are retrained on updated training data and after each iteration, the stopping criteria are checked to see whether more iterations are required. So far, there have been different AL approaches proposed, which differ in the strategies to choose the most informative data. Generally, they can be divided into *single-model-based* and *committee-based AL*. Single-model-based methods recognize the most informative data, which can reduce uncertainty, expected error rate, or output variance of the used single model [Settles, 2009]. In contrast, the committee-based approach gets the most informative data by selecting those data that

result in the most disagreement in the votes of a committee of multiple classifiers [Sebastian et al., 1992].

Evaluation: After the classification step, the matching results are obtained and the actual ER process is completed. However, users that need the ER results or researchers want to know how reliable the results are. In this scenario, the evaluation step is taken to assess the ER result quality, i.e., the effectiveness of the ER process. A detailed introduction of concepts and metrics used for evaluating ER effectiveness will be given in Section 2.2.

2.1.3 Parallel Entity Resolution

As mentioned in the described blocking step of Section 2.1.2, pair-wise ER is a time-consuming task due to its quadratic time complexity. For instance, an input dataset with 1 million records corresponds to 500 billion times of pair-wise comparisons. Suppose that one comparison takes one microsecond, then we would need 11.6 days to complete this ER task [Getoor and Machanavajjhala, 2013]. Nowadays, as the data volume keeps increasing, sequential processing to complete an ER task is no more feasible, even though blocking can be used to alleviate the problem. Therefore, the increasing data volume nowadays and the data expansion inherent to the pair-wise approaches make the use of parallel computing necessary to solve pair-wise ER tasks. For the sake of simplicity, we refer to ER that uses parallel techniques as parallel ER, otherwise, we call it serial ER.

The typical workflow of parallel ER approaches also follows the five steps of a serial ER introduced in Section 2.1.2. What makes parallel ER different from the serial ER is that in some or all steps of the ER process, parallel techniques are used to speed-up ER and achieve scalability. There are two kinds of possible parallelisms for ER:

- *Intra-step parallelism* refers to a special form of data parallelism (as opposed to task parallelism) in this context. In parallel data processing, independent operations of the same type are carried out simultaneously on elements of one data set. In this way, the large-scale data can be divided into smaller subsets and the processing problem can be solved in parallel. This form is especially suitable for problems that can be broken down into many separate, independent operations on vast quantities of data. ER is such a problem, because operations like block assignment, comparisons, etc., are often independent for records or pairs and can be performed in parallel.

- *Inter-step parallelism* is a case of task parallelism, where each step in the process can be considered as a task and can be executed in parallel. Strictly speaking, relationships between adjacent steps are not independent, the output of the previous step may be the input of its next step. Therefore, the task parallelism in ER is limited. However, some approaches support a pipelined processing pattern: because the input data volume is large after one step has completed some partial results, the next step can begin to process these intermediate results without waiting for complete results from the previous step, so a certain level of task parallelism can be achieved.

As reported in our survey of Chapter 3, the vast majority of research on parallel ER focuses only on intra-step parallelism. Therefore, in this thesis, without a specific statement, parallel ER means the intra-step parallel ER.

After introducing the preliminary knowledge for ER, next we will introduce the evaluation metrics used in the experiments of our thesis.

2.2 Evaluation of Entity Resolution Approaches

In this section, we introduce the metrics used to evaluate the proposed approaches for our research. As mentioned in Chapter 1, our research covers different aspects of ER. Therefore, different parts of our research are oriented to different targets and are evaluated by different metrics. We first introduce effectiveness related metrics in Section 2.2.1, because it is the foundation of an ER solution. Then in Section 2.2.2, efficiency and scalability related metrics are introduced.

2.2.1 Effectiveness

Effectiveness is the intrinsic goal of ER solutions, which assess the result quality. It is not only used to evaluate our research, but also the object of the evaluation step of a common ER process. There are two kinds of pair-wise ER results based on the classification step - matching and non-matching pairs. Based on the reality or the truth that a pair refers to an identical real-world entity or not, the two kinds of ER results can be classified into four categories [Hand and Christen, 2018]:

- *False Positives (FPs)* are record pairs, which are classified as matching pairs by the ER process but actually refer to different real-world entities.
- *False Negatives (FNs)* are record pairs, which are classified as non-matching pairs by the ER process but actually refer to the same real-world entity.
- *True Positives (TPs)* are record pairs, which are classified as matching pairs by the ER process that are real duplicates.
- *True Negatives (TNs)* are record pairs, which are classified as non-matching pairs by the ER process and also indeed refer to different real-world entities.

The numbers of the former three categories are used to calculate the *precision, recall* and *F-measure*, which are the common metrics to evaluate the ER effectiveness [Makhoul et al., 1999].

- *Precision* is used to reflect what percentage are true matching pairs from all classified matching pairs and is calculated as follows:

$$Precision = \frac{TP}{TP + FP} \tag{2.5}$$

- *Recall* is used to indicate how much percent of true matching pairs are correctly recognized as matching pairs and is calculated as follows:

$$Recall = \frac{TP}{TP + FN} \tag{2.6}$$

A more effective approach means higher precision and recall. However, it is actually a trade-off problem between precision and recall, thus

- *F-measure* is usually defined for effectiveness and is the harmonic mean between *precision* and *recall*:

$$F\text{-}measure = \frac{2 * Precision * Recall}{Precision + Recall} \tag{2.7}$$

The research in Chapter 6 und Chapter 7 aims to improve effectiveness and is evaluated by reporting F-measures obtained in the experiments. Particularly, the HeALER approach aims to achieve reasonable effectiveness with less labeled training data. Therefore, achieved F-measures with different number of labeled training data are shown.

2.2.2 Efficiency and Scalability

Efficiency and scalability are closely related. Efficiency, typically refers to *runtime* efficiency. It indicates how good or sufficient is the amount of time for completing an ER task, in fulfilling application requirements. Hence, in our research, the runtime for completing an ER process is reported as a metric to evaluate efficiency.

Scalability indicates the capability of a system to process changing data volumes with the same response time, for instance by flexibly adjusting processing resources [Sun and Ni, 1993]. To provide scalability, instead of serially performing an ER process, parallel techniques have been widely used for ER, which use an n-machines-cluster to parallelize the ER process. Ideally, with an n-machines-cluster the runtime of ER can be reduced n times, correspondingly, n times *speed-up* can be achieved compared to the sequential computation with only one single machine. However, in reality, it is almost impossible to achieve such an ideal speed-up due to unavoidable setup and communication overheads, uneven load distribution between machines, and other factors. Therefore, in our research, we increase the number of used nodes in a cluster and record the corresponding runtime. Then the speed-up is calculated with recorded runtime to reflect efficiency and scalability.

The above-mentioned metrics are used in the research of Chapter 4 and Chapter 5. Specially, for our research relating to load balancing strategies in Chapter 5, we also employ several other metrics. First, the imbalance ratio IR is used to show how balance each reduce partition load is adjusted with a strategy, which is calculated as follows (n is the number of reduce partitions):

$$IR = \frac{max_{i=1}^{n} \#Records_i}{\frac{\sum_{i=1}^{n} \#Records_i}{n}} \tag{2.8}$$

Second, the overhead to support the strategies to balance the reduce partition workload is also recorded. Smaller overhead is more efficient for a load balancing strategy. Third, to evaluate the memory consumption of a load balancing strategy during the pair-wise comparison step, we also represent the median gabage collection (GC) time required. Last, to evaluate the robustness of a load balancing strategy against different skew levels, we use datasets with different skew factors, which are generated

based on the same base dataset with the same number of records and blocks. It makes no sense to compare the total runtime recorded for datasets with different skew factors because although the number of records and blocks is the same, the number of comparisons differs much due to different skew factors. Therefore, we calculate the runtime of each strategy for completing one million record pairs as metrics to evaluate their performance of handling skewness.

2.3 Tools Used in the Thesis

In this section, we introduce the tools we used to support our research. First, in Section 2.3.1, we focus on the big data processing framework Apache Spark, which is heavily involved in our research. Then in Section 2.3.2, we introduce the data generator GeCo, which we used to generate the synthetic datasets for our experiments.

2.3.1 Apache Spark

Big data processing frameworks are usually open-sourced and provide straightforward programming models to ease distributed programming. Therein, Hadoop MapReduce can be counted as the first generation of computation engine for big data processing frameworks. It is simple and specializes in batch processing, but at the same time with some drawbacks. Its process has to be abstracted with "map" and "reduce" phases and before the "reducer" phase, there must be a "map" phase. Besides, it only supports disk-oriented computation and all its intermediate results have to be stored on disk, thus I/O costs limit its speed to a great extent. To overcome those drawbacks, Apache Spark has been developed and is one of the most popular frameworks nowadays. Compared to MapReduce, its programming model is more flexible without the need for abstraction to "map" and "reduce" phases and it could provide high speed by supporting in-memory computation without storing intermediate results to disk. Also, it integrates with several libraries such as Spark SQL and MLlibs, making it possible to express ER in terms of relational databases and machine learning [Zaharia et al., 2016]. Next, Apache Spark will be introduced in detail.

Spark Core: Apache Spark is designed for processing large-scale data fast for broad applications. It supports acyclic data flow and in-memory computing, which make Spark run programs up to 100 times faster than Hadoop MapReduce in memory or 10 times faster on disk [Zaharia et al., 2016]. Its main abstraction is Resilient Distributed Data (RDD), which is a collection of data partitioned across the nodes of the cluster and can be operated on in parallel, supports in-memory computing, and provides fault tolerance. RDD supports two kinds of operations: transformations and actions. Transformations only create new RDDs from existing RDDs, while actions run a computation on RDDs and return values. To enable Spark to run more efficiently, transformations in Spark are all lazy, which means transformations for datasets are only recorded but not computed right away. Besides, persisting data in memory is one of the most important capabilities in Spark. There are several persistence levels available, such as MEMORY_ONLY, MEMORY_ONLY_SER, MEMORY_AND_DISK, which differentiate them from the location to persist data (memory or disk) and whether to serialize data before persisting.

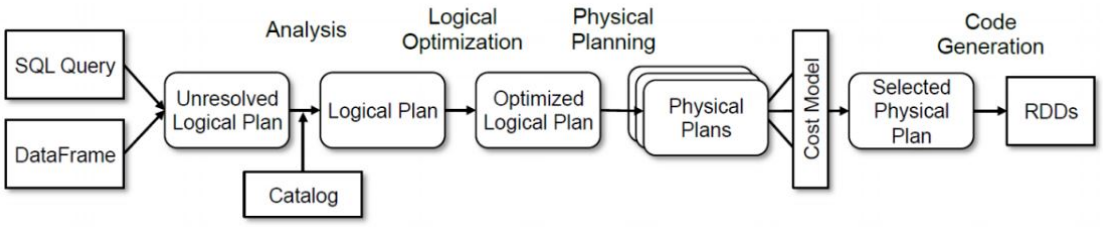


Figure 2.4: Phases of query planning in Spark SQL [Armbrust et al., 2015].

Spark Libraries: Spark also integrates four libraries: Spark SQL, MLlib for machine learning, GraphX for graphs and graph-parallel computation, and Spark streaming for building scalable fault-tolerant streaming applications [Zaharia et al., 2016]. In our research, Spark SQL and MLlib have been adopted to support our research, which will be described next.

- *Spark SQL* is the module of Apache Spark for structured data, which enables people to query structured data inside Spark programs, using either SQL or a familiar Datasets API. It does not force users to decide for a relational or a procedural API but enables users to mix both of them [Armbrust et al., 2015]. It provides the possibility to have logical and physical optimizations before the real execution. Figure 2.4 shows the four phases using its cost-based optimizer Catalyst to optimize the application. It first analyzes a logical plan from references and optimizes it, then it chooses the best physical based on costs, and last generates code to compile the query to Java code [Armbrust et al., 2015]. Besides, it also has columnar storage called Tungsten and can use Kryo serialization to replace traditional Java serialization to minimize storage cost and improve efficiency [Karau and Warren, 2017]. All the above-introduced features make Spark SQL a promising option to process structured or semi-structured data.

- *Spark MLlib* is the ML library for Spark. It implements the standard learning algorithms of classification, regression, collaborative filtering, clustering, and dimensionality reduction and makes them run fast and scalable [Meng et al., 2015]. Besides, MLlib provides pipeline APIs to combine multiple algorithms to process and learn from data.

Three Java APIs in Apache Spark: Before Spark 1.6, only RDD API and DataFrame API exist. Theoretically speaking, compared to RDD API, DataFrame API is able to run logical and physical optimizations before the real execution, but it has no type-safety for analysis errors, i.e., analysis errors cannot be caught during compile time. To overcome this shortcoming and at the same time keep the advantage of those optimizations, the Dataset API was introduced in Spark 1.6. Therefore, there are three kinds of APIs after Spark 1.6. In Spark 2.0, two structured APIs: the DataFrame API and the Dataset API are unified to a single Dataset API, and the DataFrame API is named Dataset<Row>, and the return type of a SQL query or SQL-based API is Dataset<Row> [Zaharia et al., 2016].

2.3.2 Data Generator: GeCo

Our synthetic data used in our research is generated by a data generator called GeCo [Tran et al., 2013], which is based on real-world data and can follow similar characteristics to the real data [Christen and Vatsalan, 2013]. It consists of GEnerator and COrruptor, which is specifically designed for generating ER datasets. First, the generator is responsible for generating original data, whose attribute values are based on frequency look-up files and functions provided by users. Thus, it also takes into account compounding attributes, because in reality values of some attributes depend upon values of other attributes. Afterward, the corruptor modifies some attribute values of already generated original data from the first step to generate duplicate records. The number of both original and duplicate records can be set according to evaluation requirements and so can be the maximum number of duplicates per record and the probability distribution of duplicates. It is also possible to limit the maximum number of modifications per attribute or record. For modification itself, the following ways are possible: inserting, deleting, substituting a character with another character from a specified range, and transposing two adjacent characters [Tran et al., 2013]. In this way, a synthetic dataset with original records and their duplicate records is generated. For each record, an identifier is assigned, which can show the ground truth of the dataset. Then a record generated by GeCO contains personal information with the following 14 attributes: rec-id, gender, given-name, surname, postcode, city, telephone-number, credit-card-number, income-normal, age-uniform, income, age, sex, and blood-pressure.

3. Overview and Classification of Parallel Entity Resolution Approaches

This chapter is based on the Open Journal of Big Data paper "Cloud-Scale Entity Resolution: Current State and Open Challenges" [Chen et al., 2018b]. Since the paper was published in 2018 and we reviewed only the research approaches for parallel Entity Resolution (ER) till the end of 2017. In this chapter, we re-conducted the systematic literature review process for papers published till the end of 2019 and integrated discussions for the newly-added papers. Furthermore, open challenges have been updated as well based on the updated state of parallel ER research.

As described in the previous chapters, ER is a quite time-consuming task. Facing large-scale data, the runtime of ER is often not acceptable when it is serially performed. Therefore, parallel computation has been exploited to improve efficiency and scalability. The research on parallel ER has reached a prosperous stage and many approaches have been proposed. In this chapter, we conducted a Systematic Literature Review (SLR) on parallel ER and present our results. In Section 3.1, we first give an overview of state-of-the-art approaches based on the results of our selected papers in our SLR. The methodology of our SLR to obtain the final paper set is described in Section A.1. Subsequently, in Section 3.2 we classify and compare a corpus of 58 articles based on three sets of extracted criteria, conclude solutions for efficiency-related criteria, which are the critical topics for parallel ER. At last, we expose the latent research directions for parallel ER in Section 3.3. In Section 3.4, we discuss related classifications and surveys for ER. We conclude our findings in Section 3.5.

Table 3.1: Classification based on the programming model used (1).

ID	Publication Information	Parallel Framework
P-Swoosh	Kawai et al., 2006	Parallel DBMS
Parallel linkage	Kim and Lee, 2007	Parallel DBMS
D-Swoosh	Benjelloun et al., 2007	Parallel DBMS
FERAPARDAF	Santos et al., 2007	Parallel DBMS
Febrl	Christen, 2008	Parallel DBMS
Iterative DDG	Herschel et al., 2012	Parallel DBMS
Partition-based	Jiang et al., 2013	Parallel DBMS
Pairwise document	Elsayed et al., 2008	Hadoop MapReduce
SSJ-2R	Baraglia et al., 2010	Hadoop MapReduce
Dedoop	Kirsten et al., 2010; Kolb and Rahm, 2013; Kolb et al., 2011a,b, 2012a,b,c, 2013	Hadoop MapReduce
VCL	Vernica et al., 2010	Hadoop MapReduce
MapDupReducer	Wang et al., 2010	Hadoop MapReduce
MD-Approach	Dal Bianco et al., 2011	Hadoop MapReduce
V-SMART-Join	Metwally and Faloutsos, 2012	Hadoop MapReduce
MRSimJoin	Silva and Reed, 2012; Silva et al., 2012	Hadoop MapReduce
DC2B	Kim and Shim, 2012	Hadoop MapReduce
ZkNN	Zhang et al., 2012	Hadoop MapReduce
DAA	Luo et al., 2012	Hadoop MapReduce
LINDA	Böhm et al., 2012	Hadoop MapReduce
KNN Distance filtering	Lu et al., 2012	Hadoop MapReduce
Graph-based	Kardes et al., 2013	Hadoop MapReduce
MR-DSJ	Seidl et al., 2013	Hadoop MapReduce
MultiKeyBalancing	Hsueh et al., 2014	Hadoop MapReduce
PHiDJ	Fries et al., 2014	Hadoop MapReduce

3.1 Overview of State-of-the-art Approaches of Parallel ER

With the procedures described in Section A.1, we had finally 58 papers for the subsequent classification and analysis in the area of parallel ER. In this section, we give an overview on the 58 papers.

As explained in Section 2.1.3, what differs between serial ER and parallel ER is that in each step of parallel ER data processing is tried to be parallelized instead of sequentially processed and at the same time, a certain level of task parallelism can be achieved. Regarding the first form of parallelism: data (intra-step) parallelism introduced in Section 2.1.3, ideally the speedup in run-time is equal to the number of total processors or nodes. However, ER cannot achieve full data parallelism. On the one hand, this can be caused by intrinsic reasons of parallel data processing,

Table 3.2: Classification based on the programming model used (2).

ID	Publication Information	Programming Model
Cluster Join	Das Sarma et al., 2014	Hadoop MapReduce
Graph-parallel	Malhotra et al., 2014	Hadoop MapReduce
Mass Join	Deng et al., 2014	Hadoop MapReduce
DCS++ MR-ER	Gomes Mestre and Pires, 2013; Mestre et al., 2015	Hadoop MapReduce
Sort-Map-Reduce	Ma and Yang, 2015; Ma et al., 2015	Hadoop MapReduce
SAX	Ma et al., 2016	Hadoop MapReduce
FACET	Yang et al., 2016	Hadoop MapReduce
SJT-based	Liu et al., 2016	Hadoop MapReduce
High velocity streams	Benny et al., 2016	Hadoop MapReduce
Dis-Dedup	Chu et al., 2016	Hadoop MapReduce
Meta-blocking	Efthymiou et al., 2015, 2017; Papadakis et al., 2017	Hadoop MapReduce
Sampling-based	Chen et al., 2017	Hadoop MapReduce
MSJL	Sohrabi and Azgomi, 2017	Hadoop MapReduce
KNN - DP	Zhao et al., 2017	Hadoop MapReduce
FS-Join	Rong et al., 2012, 2017	Hadoop MapReduce
Grid-based	Jang and Chang, 2018	Hadoop MapReduce
ER of healthcare data	Pita et al., 2015	Apache Spark
DCS++ Spark-ER	Mestre et al., 2017	Apache Spark
RDD-based ER	Chen et al., 2015	Hadoop MapReduce or Apache Spark
DDUB	Dou et al., 2019	Hadoop MapReduce or Apache Spark
SAVD	Rong et al., 2019	Hadoop MapReduce or Apache Spark

such as requiring time for the initialization and the communication between the processors to exchange intermediate results, or load imbalance between nodes. On the other hand, not all steps of the ER process can be parallelized perfectly, which may lower the parallel efficiency. Through our SLR, finally, we get a corpus of 58 papers. To be noted that almost all publications on parallel ER focus on data (intra-step) parallelism. Only [Santos et al., 2007] presents an approach to parallelize the processing within as well as across different steps. Therefore, our general discussion and classification in this survey are mainly data (intra-step) parallelism. Particular descriptions will be given explicitly if task (inter-step) parallelism is involved.

The 58 papers are listed in Table 3.1 and Table 3.2. We merged research presented in different papers in one row if they are done by the same researchers or the same research group, and presented methods are tightly related and can be combined. For instance, eight papers in the final literature set are regarding the research on Dedoop and we combined all eight papers into one row and named it Dedoop. This way we

have in total 45 rows in Table 3.1 and Table 3.2. For simplicity, we assign each publication a short and meaningful identifying name to simply represent them throughout this survey. Then we classify them according to the big data processing frameworks they use to implement the parallel ER: parallel DBMS, i.e., no big data processing framework used, Hadoop MapReduce-based, and Apache Spark-based. These are the currently most often used approaches, in real-world-applications as well as in academic research. Therefore, we focus on them in this overview. Nevertheless, new parallel frameworks are a topic of ongoing research and their applicability and usage for ER must be covered by future research, as discussed in Section 3.3. As we can see from Table 3.1 and Table 3.2, seven of the research do not employ a parallel framework but use Parallel DBMS to execute parallel ER tasks. 32 approaches employ only Hadoop MapReduce, and the remaining three approaches are only Spark-based. The approach presented in [Chen et al., 2015; Dou et al., 2019; Rong et al., 2019] implemented parallel ER with both Hadoop MapReduce and Apache Spark.

As we can see from the result, only 7 out of 45 approaches implemented general parallelism, i.e., using parallel DBMS. The majority of them are mostly early works, but it is certainly more relevant for real-world applications than the proportion of academic research suggests. After the programming model MapReduce became popular after the year of 2008 for parallel computing and with the support of its open-source implementation of Apache Hadoop, the vast majority of research approaches (more than two-thirds of the approaches in Table 3.1 and Table 3.2) employed it. The reasons behind are two folds: On the one hand, MapReduce provides users a simple model to express relatively sophisticated distributed programs [Pavlo et al., 2009]. Users only need to care about the implementation of the map and reduce functions, with no need to consider the partitioning of the input dataset, scheduling the program across machines, handling failures, and managing inter-machine communication. On the other hand, using parallel DBMS has some shortcomings for small and medium-sized enterprises. Parallel DBMSs are expensive [Stonebraker et al., 2010], cannot operate in a heterogeneous environment, and have very limited fault tolerance [Abouzeid et al., 2009]. Furthermore, they require high maintenance and administration effort. In academia, researchers may encounter problems, such as limited budgets and a heterogeneous environment. This may be another reason, why open-source frameworks are disproportionately popular there. Furthermore, their high level of abstraction might help researchers to concentrate on application-specific – in this case ER – research questions. Besides, 5 out of 45 approaches employed Apache Spark, which represents the current state of ER in academic research quite well regarding proportion. Spark-based techniques have rarely been applied before 2015, mainly because Spark has become an open-source Top-Level Apache Project since February 2014[1]. Three approaches from the five Spark-based approaches implemented their methods with Hadoop MapReduce as well to evaluate and compare to Spark-based ER.

As mentioned above, almost all approaches only parallelized the pair-wise-comparison step, while [Santos et al., 2007] parallelized data processing in each step and also performed different steps in parallel through its run-time system - Anthill. Besides, among publications that implement data parallelism, most of them considered only

[1] https://en.wikipedia.org/wiki/Apache_Spark

Table 3.3: General classification of parallel DBMS entity resolution.

ID	Operation Type	#Input Sources	Input Data Type
P-Swoosh [Kawai et al.]	Entity resolution	Not described	Records
Parallel linkage [Kim and Lee]	Entity resolution	2	Records
D-Swoosh [Benjelloun et al.; Kawai et al.]	Entity resolution	Not described	Records
FERAPARDAF [Santos et al.]	Deduplication	1	Records
Febrl [Christen]	Deduplication	1	Records
Iterative DDG [Herschel et al.]	Deduplication	1	Graph
Partition-based [Jiang et al.]	Similarity join	1	Strings

inter-processor parallelism. Approaches presented in [Böhm et al., 2012; Dal Bianco et al., 2011; Jiang et al., 2013; Malhotra et al., 2014] considered both intra- and inter-processor parallelism. The benefit of considering intra-processor parallelism was mentioned in [Dal Bianco et al., 2011], i.e., the communication overhead is low without excessive data copying.

3.2 Classification Based on Three Sets of Criteria

To overview and classify existing approaches we extract three sets of criteria for parallel ER, which are introduced in the following:

- The first set of criteria describes *general aspects* which are mostly application-driven including operation type, number of input sources, and input data structure.
- The second set of criteria is related to *effectiveness*, which include data pre-processing, similarity function, match mode, and clustering.
- The last set of criteria pertain to *efficiency*, which is the most important part for parallel ER and includes specific aspects of the programming model used, blocking approaches, data partitioning, load balancing, and redundancy handling.

On the one hand, these criteria can be used to compare and classify existing approaches regarding parallel ER. On the other hand, when developers are designing their own parallel ER, these criteria can help them to discuss and to weigh specific requirements of their parallel ER application. In the rest of this section, we will introduce the three sets of criteria in detail and present the classification based on those directly after each set of criteria. Therein, we also conclude the solutions for efficiency-based criteria, because they are the most critical ones for parallel ER.

3.2.1 General Criteria and Their Classification

The three general criteria focus mainly on application aspects, i.e., how and for what purpose the approaches are being used. The criteria are:

Table 3.4: General classification of MapReduce-based entity resolution (1).

ID	Operation Type	#Input Sources	Input Data Type
Dedoop [Kolb et al.]	Entity resolution	1 or n	Records
LINDA [Böhm et al.]	Entity resolution	n	Graph
Graph-based [Kardes et al.]	Entity resolution	Not described	Records
DCS++ MR-ER [Gomes Mestre and Pires]	Entity resolution	1	Records
Graph-parallel [Malhotra et al.]	Entity resolution	Not described	Records
High velocity streams [Benny et al.]	Entity resolution	2	Records
Meta-blocking [Efthymiou et al.]	Entity resolution	2	Records
Dis-Dedup [Chu et al.]	Deduplication	1	Records
Sort-Map-Reduce [Ma and Yang]	Deduplication	1	Records
MapDupReducer [Wang et al.]	Deduplication	1	Documents
MD-Approach [Dal Bianco et al.]	Deduplication	1	Records

- *Operation Type:* As mentioned above, deduplication is a special case of entity resolution, indicating that a reconciliation of found matches is an intended part of the process. Similarity joins are tightly related to entity resolution in the database domain for finding matching pairs. Although they are quite similar to ER, normally they have their particular focus points. In this survey, related research approaches, such as similarity joins and deduplication, are also included. Therefore, this criterion is set to clarify the precise operation type for each approach.

- *Number of Input Sources:* Current approaches can also be classified based on the number of input sources for ER. It is not difficult to extend ER within one single source to multiple sources. However, the extension may become complicated, not intuitive, and we may lose the advantage of knowing that the data is from two sources. Therefore, it is necessary to consider which algorithms or techniques can be used for ER among multiple sources and which can only be used within one single source. If the state is the latter one, i.e., only within one single source, they should be extended to the situation among multiple sources as needed in the future work.

- *Input Data Type:* For existing publications, the type of input data varies. In most cases, approaches do not limit the types of input data. We use "records" to represent this case because using flatly structured data units as in relational databases is the standard case considered. However, in some research, more or less complex data types are considered, e.g. documents or strings. In particular, in some research, to improve the effectiveness of ER, they consider

not only records and entities themselves but also relationships between records and entities. In this case, the input data type is an entity graph.

Based on these three criteria, we have the first set of overview tables in Table 3.3, Table 3.4, Table 3.5 and Table 3.6, which provide an overview for these criteria, once again grouped for parallel DBMS ER (Table 3.3), MapReduce-based ER (Table 3.4 and Table 3.5) and Spark-based ER (Table 3.6). To be noticed that, for the three approaches in [Chen et al., 2015; Dou et al., 2019; Rong et al., 2019], which are implemented with both Hadoop MapReduce and Apache Spark, they are put into the tables for Spark-based ER, because the implementation with Hadoop MapReduce is mostly for the purpose of evaluating Spark-based ER. The same case applies to the later effectiveness and efficiency criteria.

As can be seen in these three overview tables regarding the **operation type**, half of the publications considered a general entity resolution problem. Some of them consider only the situation that there is a single input source, which indicates a deduplication problem. The other half of publications are regarding the similarity join, in which 6 out of 24 approaches belong to the kNN similarity join. For general ER scenarios, the similarities of all candidate pairs are calculated, then a classification step is used to distinguish match or non-match pairs, which are the results for ER. In contrast, in similarity joins records from input sources, whose similarities are higher than the preset threshold, are joined together. It does not include an explicit classification step, and the dissimilar records are directly dropped and not a part of the results. Particularly, a kNN similarity join joins each record with its k most similar records.

Regarding the **number of input sources**, about half of the approaches considered the situation that the input data may be from different sources. However, most of them handled this problem only by combining multiple sources into one source. Especially, in parallel linkage [Kim and Lee, 2007], the emphasis is on studying different algorithms for the sake of better performance of the ER task, rather than combining multiple sources into one source considering whether there are duplicates in each input source or not.

At last, most of the publications did not limit the **input data type**. Jiang et al. [2013], Rong et al. [2017] and Deng et al. [2014] only consider input data with strings. Baraglia et al. [2010], Elsayed et al. [2008] and Wang et al. [2010] tried to resolve document matches. ER of health-care data [Pita et al., 2015] focuses on health data due to their specific application area. Many (kNN) similarity joins including [Chen et al., 2015; Fries et al., 2014; Jang and Chang, 2018; Luo et al., 2012; Ma et al., 2016; Rong et al., 2019; Zhang et al., 2012; Zhao et al., 2017] focus on multi-dimensional data. Furthermore, Iterative DDG [Herschel et al., 2012] and LINDA [Böhm et al., 2012] first form an entity graph, whose edges represent known relationships between records, and then use this entity graph as input data for ER. Though only these mentioned approaches consider specifics of the given application domain, this inclusion of knowledge about the data characteristics during blocking, similarity calculation, etc. can, in general, be very beneficial for overall goals like effectiveness and efficiency.

Table 3.5: General classification of MapReduce-based entity resolution (2).

ID	Operation Type	#Input Sources	Input Data Type
Pairwise document [Elsayed et al.]	Similarity join	1	Documents
SSJ-2R [Baraglia et al.]	Similarity join	1	Documents
VCL [Vernica et al.]	Similarity join	1 or n	Records
V-SMART-Join [Metwally and Faloutsos]	Similarity join	1	Sets, multisets and vectors
MRSimJoin [Silva and Reed]	Similarity join	2	Records
MR-DSJ [Seidl et al.]	Similarity join	1	Vectors
PHiDJ [Fries et al.]	Similarity join	1	High dimensional vectors
Cluster Join [Das Sarma et al.]	Similarity join	1 or n	Records
DAA[Luo et al.]	Similarity join	1	High dimensional vectors
Mass Join [Deng et al.]	Similarity join	1 or 2	Strings
SAX[Ma et al.]	Similarity join	2	High dimensional vectors
FACET [Yang et al.]	Similarity join	1 or 2	Vectors
SJT-based [Liu et al.]	Similarity join	1	Records
Sampling-based [Chen et al.]	Similarity join	2	Records
MSJL [Sohrabi and Azgomi]	Similarity join	2	Records
FS-Join[Rong et al.]	Similarity join	1	Srings
Grid-based[Jang and Chang]	(KNN) Similarity join	2	Multi - dimensional data
KNN - DP[Zhao et al.]	KNN Similarity Join	2	Multi - dimensional data
DC2B[Kim and Shim]	KNN similairty join	1	Records
ZkNN[Zhang et al.]	KNN similarity join	2	Multi - dimensional data
KNN Distance filtering[Lu et al.]	KNN similarity join	1 or 2	Records

Table 3.6: General classification of Spark-based entity resolution.

ID	Operation Type	#Input Sources	Input Data Type
DDUB [Dou et al.]	Entity resolution	1 or 2	Records
ER of healthcare data [Pita et al.]	Entity resolution	n	Health data
DCS++ Spark-ER [Mestre et al.]	Entity resolution	1 or 2	Records
SAVD[Rong et al.]	Similarity join	1 or 2	High dimensional data
RDD-based ER [Chen et al.]	KNN similarity join	1	Multi - dimensional data

3.2.2 Effectiveness-Related Criteria and Their Classification

This subsection presents criteria related to effectiveness and the classification of approaches based on them. The following criteria are considered:

- *Data preprocessing:* This criterion indicates whether there are any data preprocessing steps in algorithms or techniques of the approaches and which kinds of procedures are used. With proper data preprocessing steps, noises, which can negatively affect the effectiveness of the subsequent similarity calculation and classification steps, can be reduced. Therefore, we consider data preprocessing as an effectiveness-related criterion.

- *Similarity function:* The reason for considering this criterion for classifying parallel ER techniques is that some approaches apply parallel techniques only suitable when specific similarity functions are used. Thus, publications should be pointed out where specific similarity functions are part of the core procedure or whether their algorithms or techniques can be applied independently.

- *Match Mode:* This criterion addresses another aspect of the pair-wise comparison step. It does not concern the specific function used for comparison, but after a pair of records is identified by a similarity function as a match, what kinds of further steps will be done to improve the results, e.g. whether matching records are merged or if matching results are propagated.

- *Clustering:* Applications can also be classified according to whether they cluster matching records based on pair-wise comparison results and which clustering methods they use.

Table 3.7, Table 3.8, Table 3.9, Table 3.10 and Table 3.11 present the classification of publications based on these four effectiveness related criteria.

Regarding data **preprocessing**, half of the approaches support preprocessing steps in one way or the other, among which cleaning and standardization are the most commonly used techniques. The approaches in [Christen, 2008; Santos et al., 2007;

Table 3.7: Effectiveness consideration of parallel DBMS entity resolution.

ID	Data Preprocessing	Similarity Function	Match Mode	Clustering
P-Swoosh [Kawai et al.]	No preprocessing	Not described	Merge after match	No clustering
Parallel linkage [Kim and Lee]	No preprocessing	S-cc; s-dcself; s-dd1; s-dd2; s-dd3	Merge after match	No clustering
D-Swoosh [Benjelloun et al.]	No preprocessing	Not described	Merge after match	No clustering
FERAPARDAF [Santos et al.]	Standardization; cleaning	Not described	No merge	No clustering
Febrl [Christen]	Hidden-markov-model for cleaning and standardization	Matching attribute weights	No merge	No clustering
Iterative DDG [Herschel et al.]	Entity graph as input	Self-defined similarities	Iterative and propagate	No clustering
Partition-based [Jiang et al.]	Partition and substring comparison	Extension-based verification	No merge	No clustering

Wang et al., 2010] also supported data cleaning and standardization. Christen [2008] suggested a three-step data cleaning and standardization based on hidden Markov models [Christen, 2008]. As a preparation to special blocking methods used in some publications, the data preprocessing step in [Deng et al., 2014; Rong et al., 2012, 2017, 2019; Vernica et al., 2010; Wang et al., 2010] include transforming input records to tokens and [Jiang et al., 2013] first partitions the input records (records here are all strings) and then transforms input strings into substrings. The approaches presented in [Böhm et al., 2012; Herschel et al., 2012] are graph-based and consider relationships between input records. Therefore, they build an entity graph during the data preprocessing step. Besides, in [Kardes et al., 2013], Soundex or a phonetic algorithm is used to preprocess the input data. For research approaches, whose focuses are handling reducer skew, sampling, or other procedures are required to get data distribution statistics. For research approaches, whose input data is multi-dimensional vectors, dimension reduction procedures are often applied.

Similarity functions used in each publication vary, as all the different approaches choose suitable functions for their own scenarios. Since in publications on parallel ER the specific similarity functions used are often not a focus, around half of the approaches did not discuss the specific similarity function(s) they used. Dou et al. [2019] focused their research only on blocking algorithms and no related information on later steps is provided. Kim and Lee [Kim and Lee, 2007] developed a series of algorithms for ER among two sources by considering whether sources are

Table 3.8: Effectiveness consideration of MapReduce-based entity resolution (1).

ID	Data Preprocessing	Similarity Function	Match Mode	Clustering
Pairwise document [Elsayed et al.	Stemming; stopwords removal; Df-cut	The symmetric variant of OkapiBM25 [Olsson and Oard]	No merge	No clustering
SSJ-2R [Baraglia et al.]	Stemming; stopwords removal; normalization; lexicon extraction; sorting features	Not described	No merge	No clustering
Dedoop [Kolb et al.]	No preprocessing	Not described	No merge	No clustering
VCL [Vernica et al.]	String to prefix tokens	Coefficients; Jaccard; Tanimoto; cosine	No merge	No clustering
MapDup-Reducer [Wang et al.]	Cleaning; parsing; tokensation	Not described	No merge	No clustering
MD-Approach [Dal Bianco et al.]	No preprocessing	Not described	No merge	No clustering
V-SMART-Join [Metwally and Faloutsos]	Stopwords removal	Nominal Similarity Measures	No merge	No clustering
MRSimJoin [Silva and Reed]	No preprocessing	Various functions possible	No merge	No clustering
DC2B[Kim and Shim]	No preprocessing	Euclidean; Minkowski distance	No merge	K closest pairs
ZkNN[Zhang et al.]	Using z-value to get approximate one dimensional data	Not described	No merge	K closest pairs
DAA[Luo et al.]	Sampling; DAA	Euclidean distance	No merge	No clustering
LINDA [Böhm et al.]	Edges added to build graph	Not described	Iterative and propagate	No clustering

Table 3.9: Effectiveness consideration of MapReduce-based entity resolution (2).

ID	Data Preprocessing	Similarity Function	Match Mode	Clustering
KNN Distance filtering[Lu et al.]	Pivot selection	Not described	No merge	K closest pairs
Graph-based [Kardes et al.]	Soundex or phonetic algorithms	Feature-based	No merge	Transitive closure; SClust
MR-DSJ [Seidl et al.]	No preprocessing	Distance-based functions	No merge	No clustering
PHiDJ [Fries et al.]	no preprocessing	Distance-based functions	No merge	No clustering
Cluster Join [Das Sarma et al.]	No preprocessing	Not described	No merge	No clustering
Graph-parallel [Malhotra et al.]	No preprocessing	Not described	Merge or no merge	Connected components
Mass Join [Deng et al.]	Token-based signature generation	Jaccard; edit distance; set- and character-based	No merge	No clustering
DCS++ MR-ER [Gomes Mestre and Pires]	No preprocessing	Jaro-Winkler	No merge	No clustering
Sort-Map-Reduce [Ma et al.]	No preprocessing	Not described	No merge	No clustering
SAX[Ma et al.]	Computing PAA representations; SAX strings of each vector	Euclidean distance	No Merge	No clustering
FACET [Yang et al.]	Getting additional information to prepare for blocking	Cosine or Dice similarity(-based)	No merge	No clustering
SJT-based [Liu et al.]	No	Not described	No merge	No clustering

Table 3.10: Effectiveness consideration of MapReduce-based entity resolution (3).

ID	Data Preprocessing	Similarity Function	Match Mode	Clustering
High velocity streams [Benny et al.]	2rd boundaries; stemming; stop word removal	Average result of 13 functions	No merge	No clustering
Dis-Dedup [Chu et al.]	No preprocessing	Edit distance	No merge	No clustering
Meta-blocking [Efthymiou et al.]	Redundancy positive block collections	Not described	No merge	No clustering
Sampling-based [Chen et al.]	Centroid selection based on sampled data	Any functions possible	No merge	No clustering
MSJL [Sohrabi and Azgomi]	Building HAROT matrix for using LSH	Jaccard similarity coefficient	No merge	No clustering
KNN - DP[Zhao et al.]	Sampling; partition boundary determination	application-based	No merge	K nearest pairs
FS-Join[Rong et al.]	String to prefix tokens	Jaccard default; others possible	No merge	No clustering
Grid-based[Jang and Chang]	Sampling	L_p distance	No merge	No/K closest pairs

Table 3.11: Effectiveness consideration of Spark-based entity resolution.

ID	Data Preprocessing	Similarity Function	Match Mode	Clustering
DDUB [Dou et al.]	No preprocessing	None, focus only on blocking	No merge	No clustering
ER of healthcare data [Pita et al.]	No preprocessing	Dice or bit vectors comparison	No merge	No clustering
DCS++ Spark-ER [Mestre et al.]	No preprocessing	Jaro-Winkler	No merge	No clustering
SAVD [Rong et al.]	Z-normalization; dimension reduction	Euclidean distance	No merge	No clustering
RDD-based ER [Chen et al.]	No preprocessing	Top-k-DC	No merge	K closest pairs

clean or dirty in different scenarios. Except for the above-mentioned research, other approaches have their own similarity functions, where Jaccard, edit distance, and coefficient functions are used more commonly than others. Particularly, for resolving multi-dimensional vectors, L_p distance is often used, in which Euclidean distance is a default similarity function.

Regarding the **match mode**, most of the approaches terminate ER tasks after they have matching results for all records. In [Böhm et al., 2012; Herschel et al., 2012], similarity functions were iteratively used to improve results and all results should propagate to the whole entity graph.

Regarding **clustering** and dealing with multiple matches of single records, the majority of approaches do not cluster records after local comparisons. In [Malhotra et al., 2014] Malhotra took each connected component as a cluster. Kardes in [Kardes et al., 2013] proposed a clustering strategy called sClust to better cluster records based on the results after computing the transitive closure. Implicitly, the cluster approach used in kNN-related approaches ([Chen et al., 2015; Jang and Chang, 2018; Kim and Shim, 2012; Lu et al., 2012; Zhang et al., 2012; Zhao et al., 2017]) are considered to take the top-k closest pairs of records as a cluster.

3.2.3 Efficiency-Related Criteria and Their Classification

In this section, we will introduce the last group of criteria, which are related to efficiency, i.e., mainly focused on runtime performance aspects such as response time, throughput, and scalability. As ER is parallelized to improve especially towards these goals, their consideration is of great importance within this overview. The following four main criteria are considered for the classification:

- *Blocking:* It is a vital step to improve the efficiency of ER. Therefore, for large-scale data, it should be considered and indeed is often discussed in current parallel ER, with some research only focusing on finding efficient blocking strategies.

Table 3.12: Efficiency consideration of parallel DBMS entity resolution.

ID	Blocking	Data partitioning & Load balancing	Redundancy handling
P-Swoosh [Kawai et al.]	No or standard blocking	Master node: sliding windows and send non-match records to slave nodes; Horizontal and vertical load balancing	Not described
Parallel linkage [Kim and Lee]	Not described	Replication of source A to all processors and evenly partitioned source B	Not described
D-Swoosh [Benjelloun et al.]	No or standard blocking	Scope functions	Value equality; hierarchies; linear ordering; Reps functions
FERAPA-RDAF [Santos et al.]	Standard blocking	A labeled stream in Anthill	Not described
Febrl [Christen]	Standard blocking; sorted neighborhood; q-gram	Not described	Not described
Iterative DDG [Herschel et al.]	Entity graph as input	Evenly partition input entity graph	Not described
Partition-based [Jiang et al.]	Inverted index	Evenly partitioning	Substring selection; content filter; effective indexing

- *Data partitioning and load balancing:* For parallel ER, when parallelizing the ER process with an n-machines-cluster, an n times speed-up can be achieved compared to the sequential computation with only one single machine. However, in reality, it is almost impossible to achieve such an ideal speed-up due to unavoidable setup and communication overheads, especially due to uneven load distribution between machines. In this case, the machine with an overpopulated task will become the straggler of the whole cluster and dominate the overall runtime. Therefore, how to partition the input data, or data after defining blocking keys, allocate the partitions to available processing units (such as cores, processors, reducers) and maintain balanced load is an important research question in parallel ER. Different reasons can lead to an unbalanced load. One main type of load imbalance is due to the blocking strategy used in parallel ER. As analyzed above, it is popular to use big data processing frameworks, such as Hadoop MapReduce or Apache Spark, to implement a parallel ER process because of their simple programming model [Pavlo et al., 2009] and their applicability to the ER problem with blocking [Kolb et al., 2012a]. The default partitioning strategies used in them are range or hash partitioning, which aim to achieve a balanced load by partitioning the equal range of data to each machine (range partitioning) or spreading data based on the key (hash partitioning) [Zaharia et al., 2012]. These strategies only balance the load when the keys are evenly distributed. For parallel ER with blocking, the sizes of each block usually vary much due to the different number of records with different blocking keys. Then all records with the same blocking key are assigned as one partition to a reducer, which leads to an uneven distribution of the whole workload to available reducers. Furthermore, the pair-wise similarity comparison and processing performed on the reducers amplifies the impact of the skew. To achieve the best speed-up and scale up the ER process by combining the use of blocking and parallel computation, it is of great importance to detect possible uneven workload distributions and develop skew handling strategies to improve efficiency and scalability. Another possible load imbalance for parallel ER is due to the processing skew. Even though the number of record pairs processed by each processing unit is similar, the time required to process one pair may differ much due to the different sizes of records. This kind of load imbalance should also be considered.

- *Redundancy handling:* This criterion signifies some detailed measures to reduce the total run time, which includes reducing the number of record pairs to be compared and reducing the communication effort between different processors.

Table 3.12, Table 3.13, Table 3.14, Table 3.15 and Table 3.16 provide an overview and classification of the 35 considered approaches based on the above-explained efficiency criteria. Since data partitioning and load balancing are tightly related, they are in the tables as one column. Because the efficiency-related criteria, as a motivation of parallel ER, is the most important set of criteria, we discuss each criterion in detail and describe solutions developed for each of the involved problems in the following.

Table 3.13: Efficiency consideration of MapReduce-based entity resolution (1).

ID	Blocking	Data partitioning & Load balancing	Redundancy handling
Pairwise document [Elsayed et al.]	Inverted index	Block-based	Not described
SSJ-2R [Baraglia et al.]	Prefix filtering with inverted index	Block-based; Bucketing technique	Broadcast the remainder file
Dedoop [Kolb et al.]	Standard blocking; (multi-pass) sorted neighborhood	BlockSplit; PairRange (BDM)	Check overlapping blocking keys
VCL [Vernica et al.]	Prefix tokens or PP-join+ [Xiao et al., 2011]	3 stages (BTO/OPTO; BK/PK; BRJ/OPRJ); A Round-Robin order	Not described
MapDup-Reducer [Wang et al.]	PP-join+ [Xiao et al., 2011]	Block-based	Not described
MD-Approach [Dal Bianco et al.]	Two-step blocking with sliding windows functions	Block-based	Not described
V-SMART-Join [Metwally and Faloutsos]	Virtual inverted index	Stopword removal; dividing overloaded reducer (sharding algorithm); MapReduce combiner	Not described
MRSimJoin [Silva and Reed]	Ball partitioning in Quick-Join [Jacox and Samet, 2008]]	Base and window-pair parition	Not described
DC2B[Kim and Shim]	Safe bucket assignment	TopK-P-MR/TopK-F-MR	Not described
ZkNN[Zhang et al.]	No/R-tree	Size-based paritioning using space filling curve	Not described
DAA[Luo et al.]	No	Hash partitioning	I-DAA; OSFR; TSFR

Table 3.14: Efficiency consideration of MapReduce-based entity resolution (2).

ID	Blocking	Data partitioning & Load balancing	Redundancy handling
LINDA [Böhm et al.]	First rank pairs to assign similar pairs to a same workpackage	Workpackage-based; Server-controlled	Not described
KNN Distance filtering[Lu et al.]	Pivot-based	Voronoi diagram based	grouping strategies
Graph-based [Kardes et al.]	Two-step blocking with binomial tree structure	Block-based	Not described
MR-DSJ [Seidl et al.]	Grid-based blocking	Block-based	Smaller or equal cell ID; Bit code; MindistCell; MindistPair
PHiDJ [Fries et al.]	Grid-based blocking	Block-based; Variable grid width	All measures in MR-DSJ; Dimension group ID
Cluster Join [Das Sarma et al.]	Each partition as one block	Home and outer partitioning; Dynamic, 2d-hashing to split oversized partitions	Candidate filters; remove mapping-phase-redundancy
Graph-parallel [Malhotra et al.]	Locality sensitive hashing via Min-hash	First loading a graph to show the blocking result, then sending records to buckets with vertexes; RCP	First transferring record ID then real records
Mass Join [Deng et al.]	Standard blocking	Greedy/random strategy; Multitokens instead of single tokens	2-phase verification; merge key-value pairs; light-weight filter unit; string ids to replace strings
DCS++ MR-ER [Gomes Mestre and Pires]	DCS++ [Draisbach et al.	BlockSlicer	Transitive closure

Table 3.15: Efficiency consideration of MapReduce-based entity resolution (3).

ID	Blocking	Data partitioning & Load balancing	Redundancy handling
Sort-Map-Reduce [Ma et al.]	(Multi-pass) sorted neighborhood	Partition using preset functions	Not described
SAX[Ma et al.]	SAX representations as blocking keys	Size-based partitioning	Filtering using PAA and SAX
FACET [Yang et al.]	Prefix and length filtering	Not described	Removing duplicate pairs using key
SJT-based [Liu et al.]	SJT indexing	Extended EFM graph partitioning	Inter-node comparison pruning
High velocity streams [Benny et al.]	Weighted-graph-based blocking	Pair-based	Pruning graph of blocking
Dis-Dedup [Chu et al.]	Min-hash	Triangle distribution	Avoid comparing redundant pairs
Meta-blocking [Efthymiou et al.]	Three-stage Meta-blocking	Exploiting the power law distribution of block cardinality then evenly partitioning	Not described
Sampling-based [Chen et al.]	Each partition is a block	CPM and KPM partition methods to achieve load balancing	Range-object; double-pivot; pivot filtering; plane sweeping
MSJL [Sohrabi and Azgomi]	LSH	Band-based partitioning	Not described
KNN - DP[Zhao et al.]	LSH/z-value-based	Dynamical adjusting partitioning	Not described
FS-Join[Rong et al.]	Inverted index; pivotToSegments	Vertical and horizontal partitioning	StrL-; SegL-; Seg1-; SegD-filter; reverse ordering; record canonicalize
Grid-based[Jang and Chang]	No	(Variable-sized) grid partitioning; density based load balancing	Bit code

Table 3.16: Efficiency consideration of Spark-based entity resolution.

ID	Blocking	Data partitioning & Load balancing	Redundancy handling
DDUB[Dou et al.]	Density-based unsupervised blocking	Randomly split the dataset	Not described
ER of healthcare data [Pita et al.]	Standard blocking	Block-based	Not described
DCS++ Spark-ER [Mestre et al.]	DCS++ [Draisbach et al.]	Fixed input partition size	Transitive closure
SAVD [Rong et al.]	SAX aggregation	Vertical partitioning	Filtering to prune FPs; triangle inequality
RDD-based ER [Chen et al.]	LSH; BKDRhash function	Block-based	Spark filter

Blocking

Blocking as an efficient technique to reduce the search space is considered by most publications. Standard blocking, i.e., using a single attribute or combined/concatenated attributes as the blocking key, is most often considered because of its simplicity and efficiency. Nevertheless, because of data quality issues, this approach may decrease effectiveness, and unevenly distributed key values may lead to data skew. Therefore, other blocking methods, such as the sorted neighborhood method [Hernández and Stolfo, 1995], q-grams [Navarro et al., 2000], inverted indexes [Bell et al., 1999] and locality sensitive hashing [Indyk and Motwani, 1998], are also used in the approaches. Particularly, PP-joins are used twice in the listed approaches, which is a new blocking technique that exploits the ordering information and can drastically reduce the candidate set sizes and, hence, improve the efficiency [Xiao et al., 2011].

Except for the mentioned traditional blocking techniques applied as a single step, to solve the data skew problem, two-step blocking is considered in some approaches. Two-step blocking will be discussed as a method to solve the load balancing problem in the following.

In this overview of parallel ER, we only present an overview on choices made in existing publications for the blocking step in parallel ER, and we do not discuss the details and the performance issues for different blocking techniques, as mentioned above, [Papadakis et al., 2016] and Christen [2012a] provided more detailed discussions and evaluations for existing blocking techniques [Christen, 2012a; Papadakis et al., 2016].

Data partitioning

Most of the approaches presented did not discuss data partitioning in detail. They partition and allocate the input data randomly or they first define blocking keys,

then just assign each block to each processing unit without considering the load balancing problem, which belongs to horizontal partitioning. According to [Kirsten et al., 2010] the following three general types of horizontal partitioning strategies can be distinguished:

- *Size-based partitioning:* evenly partitioning the input data to several subsets, where the number of partitions should be smaller than the number of available nodes.
- *Pair-based partitioning:* first, all pairs that need to be compared for the next step are generated, then evenly dividing the record pairs into several subsets.
- *Block-based partitioning:* this method is designed especially for ER with blocking techniques. It distributes each block to one separate node. The method is straightforward, but it suffers from a potential load unbalancing problem, as blocks may differ in their sizes. A node with a big block will dominate the runtime and drag down the entire parallel processing.

Except for the aforementioned three general methods to partition the data to each node, Silva and Reed [2012] provided a partitioning method from a special perspective in [Silva and Reed, 2012; Silva et al., 2012]. They extended the QuickJoin ball partitioning [Jacox and Samet, 2008] to partition the input dataset iteratively until the sizes of all partitions of data fit a single node, then similarity comparison is only needed to be done within a single node. This makes the QuickJoin ball partitioning become a blocking technique at the same time. Accordingly, their partitioning method also belongs to the aforementioned block-based partitioning.

Another type of partitioning is vertical partitioning. As we can see from the tables concerning efficiency aspects of approaches, Rong et al. [2017, 2019] applies vertical partitioning, which can get rid of workload imbalance problems and data replications.

Load Balancing

As can be seen in the relevant tables, only a few of the presented approaches proposed a specific load balancing strategy, but rather focus on other implementation aspects. However, as explained above, load balancing is very important and is a factor that can significantly influence the efficiency of parallel ER.

Among the approaches aiming to provide a balanced load, the majority of them consider only the case due to the block size skew. Based on the publications referenced in the tables, as well as other load-balancing-focused publications, we point out two typical solutions for this type of imbalanced load.

- *Prevention-based methods:* This means generating blocks less than a pre-set size. Oversized temporary blocks have to be divided into several sub-blocks until all blocks have less than the pre-set size. Besides, in the presented approaches two minor techniques are used to optimize this method. One suggests using a binomial tree structure to build (sub-)blocks [McNeill et al., 2012], the other one is using a sliding window to lower the false-negative rate [Dal Bianco et al., 2011].

- *Remedying-based methods:* When the input data and applied blocking strategy lead to oversized blocks, some approaches suggest remedying this load balancing problem by two kinds of solutions. The first solution is to divide existing over-sized blocks into several sub-blocks to keep the number of comparisons of all blocks under the average reduce workload and then redivide blocks to each reducer [Kolb et al., 2012b]. This solution appears to be similar to prevention-based methods. However, since blocks have already been generated, two steps are needed for this method. First, all block sizes should be known. Then, the oversized blocks should be eliminated and very small blocks may be combined. For the first step, an accurate or approximate data structure is used to store the block distribution information. For the second step, oversized block elimination can be done in a block-based or a pair-based approach. In the block-based approach, the oversized blocks are directly divided into smaller ones. In the pair-based approach, all the needed compared pairs are calculated and evenly distributed across nodes.

  Different approaches have been developed and a short overview is provided in Table 3.17. To store the block distribution information, Kolb et al. proposed to use a so-called Blocking Distribution Matrix (BDM). However, Yan et al. pointed out that the scalability of the accurate data structure, such as BDM, is limited. Therefore, they proposed to adopt a Sketch data structure called FastAGMS sketch to approximately estimate the block size, which is scalable because of the fixed size of sketch [Yan et al., 2013a].

  To remove the bottleneck blocks, both BlockSplit [Kolb et al., 2012b] and BlockSlicer [Gomes Mestre and Pires, 2013] are block-based. However, their way to split the overpopulated blocks differs. BlockSplit considers each block as m sub-blocks based on the m map input partitions. Because the record in each sub-block needs to be compared to any other record in the same or other sub-blocks, the comparisons of the original block is then split into $\frac{1}{2} \cdot m \cdot (m-1) + m$ parts (called match tasks in [Kolb et al., 2012b]). The $\frac{1}{2} \cdot m \cdot (m-1)$ part corresponds to those match tasks, whose records are from two different sub-blocks. m comparisons are those match tasks, whose records are in the same sub-block. While BlockSlicer designs the splitting differently. It does not depend on the m input partitions. It first collects all records of an overpopulated block together, then calculates the stop record, by adding whose comparisons to the current comparisons the number of comparisons exceeds the average reduce workload. In this way, the splitting performance will not be affected by the input partitions and the number of replicated records is much smaller. After the bottleneck blocks have been removed with the above-introduced splitting approaches, both of them use a greedy load balancing strategy to assign the re-organized blocks to reducers. With the greedy strategy, BlockSplit and BlockSlicer can provide a balanced reduce load for most cases. The cell-block division approach is also block-based [Yan et al., 2013a], which is specially designed based on the cells of their sketch structure.

  The authors of BlockSplit also proposed a pair-based approach called PairRange. Compared to their BlockSplit approach, PairRange can generate a more balanced workload but its additional overhead will deteriorate the over-

Table 3.17: Classification of remedy-based load balancing strategies.

Statistic Structure	Bottleneck Blocks Removal	Proposed approaches
Accurate	Block-based	BlockSplit [Kolb et al., 2012b]
		BlockSlicer [Gomes Mestre and Pires, 2013]
	Pair-based	PairRange [Kolb et al., 2012b]
		Multiple keys [Hsueh et al., 2014]
Approximate	Block-based	Cell-block devision [Yan et al., 2013a]
	Pair-based	Cell-range devision [Yan et al., 2013a]

all execution time when the data set is relatively small [Kolb et al., 2012b]. Similarly, the authors of the cell-block division also proposed cell-range division approach, which is pair-based and also specially designed based on the cells of their sketch structure. However, due to the approximate estimation block sizes and their splitting strategies, the reduce workload imbalance cannot be ignored. Another approach considers the blocking case with multiple keys, whose bottleneck block removal strategy is pair-based. It first generates all pairs based on the result of blocking, then assigns record pairs numbers to count and mark them, last sends them to reducers in a round-robin fashion [Hsueh et al., 2014].

For the latter type of load imbalance due to processing skew, Rong et al. [2012] propose a simple approach for record attributes that are strings by considering the lengths of strings, because their similarity calculation is directly related to strings lengths. The records in each processing unit should contain similar distribution in terms of string lengths. Compared to the load imbalance caused by block skew, much less attention has been paid for the processing skew in ER, which should be more considered in future research.

Redundancy handling

In the five tables on efficiency (Table 3.12, Table 3.13, Table 3.14, Table 3.15, and Table 3.16), measures for redundancy handling are listed in the last column. When developers design a new workflow for ER, possible optimizations can be inspired by those measures, and we will classify them into four categories. Specific approaches mentioned in the tables may be useful and can be directly applied to other applications to improve the performance. We categorized those measures into the following types:

1. Measures to remove redundant comparison pairs caused by overlapping between blocks: These measures deal with redundant or unneeded comparisons due to the overlapping of blocks that are generated, where common pairs should be detected and compared only once. Existing methods to handle this redundancy can be found in the following publications: [Benjelloun et al., 2007; Das Sarma et al., 2014; Deng et al., 2014; Fries et al., 2014; Jang and Chang, 2018; Kolb et al., 2012a,c; Rong et al., 2012; Seidl et al., 2013; Yang et al.,

2016]. The shared idea to solve this problem is first identifying all candidate blocks of a record pair, then choosing the block with the smallest block ID to be responsible for comparing the pair.

2. Transitive closure: With transitive closure, a record pair can be directly identified as match or non-match without a comparison between them, if they can be deduced with two rules we explain next. The first one is the deduced match case: If we know record pairs (a,b) and (b,c) are both matching pairs, then we deduce that the pair (a,c) is also a matching pair. The second one is the deduced non-match case: If we know the record pair (a,b) is a matching pair and the other record pair (b,c) is a non-matching pair, then we deduce that the pair (a,c) is a non-matching pair. Mestre et al. [2017] have applied the transitive closure during the step of pair-wise comparison to reduce the number of record pairs that need to be compared [Gomes Mestre and Pires, 2013; Mestre et al., 2015, 2017]. To be noticed, for the similarity calculation in the vector space, the triangle inequality can be similarly employed to avoid unnecessary computations [Rong et al., 2019].

3. Further pruning techniques: There is a variety of other pruning technologies used to reduce the number of record pairs that address different aspects of the input data, processing frameworks, etc., for which the details can be found in [Benny et al., 2016; Chen et al., 2017; Das Sarma et al., 2014; Deng et al., 2014; Efthymiou et al., 2017; Fries et al., 2014; Jiang et al., 2013; Liu et al., 2016; Ma et al., 2016; Rong et al., 2012, 2019; Seidl et al., 2013].

4. Avoiding the transfer of unnecessary data: All of the above three categories are used to reduce the number of required comparisons. But redundancy may also refer to the communication effort between different processors or nodes for transferring unnecessary data. Malhotra et al. [2014] and Deng et al. [2014] introduced their measures to avoid unnecessary communication cost, which transfers record IDs instead of records themselves to reduce the overhead. Baraglia et al. [2010] used the broadcast to reduce the communication overhead.

5. Reducing shuffling cost: In parallel computation, shuffling is a very expensive operation. In some research, there are also some approaches to reduce the shuffling cost. [Rong et al., 2012] propose record canonicalization to generate a new record instead of appending all prefix tokens to the original record to reduce the shuffling cost. Lu et al. [2012] propose geometric and greedy grouping strategies to reduce the number of replicas, which can lower both shuffling and computation cost.

This list of types does indicate further optimization potential for existing approaches, through adding and studying the impact of redundancy elimination techniques. Our categorization of types may serve as a guideline to consider for adopting redundancy elimination or developing new approaches.

3.3 Open Challenges

From the descriptions in the previous sections, it is obvious that currently there are many ongoing research activities in the field of parallel ER. Partly building on established solutions, like using Parallel DBMS, partly being inspired by the availability of new parallel programming frameworks developed to support Big Data and Cloud-scale data processing. Many interesting specific solutions were developed, which often complement each other, but sometimes address contradicting requirements of divergent applications.

Even considering the presented scale of available solutions, from our point of view, several questions remain open for future solutions in parallel ER:

- *Choosing a suitable big data processing framework:* As outlined before, there was no systematic analysis of the required properties of a framework. Hadoop MapReduce was mostly used because it was popular and allowed some improvement. So far, based on our SLR results, there was only five research using Apache Spark. It is said to be able to run programs up to 100 times faster than Hadoop MapReduce in memory, or 10 times faster on disk[2]. This and other criteria indicate possible room for improvement in terms of efficiency. It also integrates SQL library, MLlib, Graph library to its core, which provides more implementation options with available low-level and high-level APIs and more convenience to do learning or graph-based research for ER. Except for Hadoop MapReduce and Apache Spark, there have been other new frameworks as well, e.g., Apache Flink [Carbone et al., 2015; Friedman and Tzoumas, 2016]. Facing these available implementation options, it needs to be studied which one is the best option needs to be studied. Although the corresponding choice may depend on the specific application scenarios, comparisons between different implementations in terms of efficiency can provide guidance to make the decision.

- *Load balancing strategies for block-skewed ER:* As briefly introduced in Section 3.2.3, the existing load balancing strategies have certain drawbacks. New load balancing strategies are required to provide stable and robust solutions.

- *Blocking techniques for large-scale data* Some familiar blocking techniques such as canopy clustering [McCallum et al., 2000], iterative blocking [Whang et al., 2009], are not deeply studied in parallel ER and new blocking techniques may be developed for large-scale data.

- *Classification:* For the classification step, learning-based approaches have been widely used. On the one hand, because the vast majority of research in parallel ER only considers threshold-based classification, learning-based classification should be more explored along with the development of machine learning libraries in distributed environments. On the other hand, learning-based classification requires sufficient training data to provide reasonable effectiveness, which is labeled by humans. The human cost is often expensive. Therefore,

[2] http://spark.apache.org/

techniques such as active learning should be more studied for ER to reduce human effort.

- *Graph-based parallel ER:* Based on our results, there is only little research on this area [Herschel et al., 2012; Kardes et al., 2013; Malhotra et al., 2014]. However, when relationships between records are available, by considering these relationships with graph-based approaches, parallel ER can benefit from it and improve the effectiveness while improving the performance by parallelism. The research on graph-based parallel ER can turn to some graph processing systems, such as GPS [Salihoglu and Widom, 2013] and Pregel [Malewicz et al., 2010]. It can benefit domains like the management of scholarly publications.

- *Learn from each other:* As we can see from the research between general ER and similarity joins, although they are quite similar, they have different focuses. ER pays much attention to the blocking step, while the research on similarity joins proposes many filtering strategies to further improve efficiency. Particularly, in kNN related research, dimension reduction has been well studied. The general ER research should consider how to handle multi-dimensional data as well, since it is more challenging than low-dimensional data.

- *Factors affecting the efficiency of parallel ER:* For serial ER, the majority of the time used for an ER process is the similarity calculation part, which is also applied to parallel ER. However, for parallel ER, to reduce the runtime for calculating similarities given a candidate pair set, a more balanced load between different nodes is the key point. Along with ever-increasing data volume, load balancing strategies are always an important topic. Besides, in a distributed environment, the shuffling cost is another important factor, which can influence the efficiency much. Existing research only slightly tackled this part by reducing the data size of shuffling. Reducer assignments by considering locality can be studied to reduce the shuffling cost.

- *Task parallelism:* No research except [Santos et al., 2007] discusses task parallelism. However, since each step needs time to process large-scale data, task parallelism is suitable for ER to reduce its entire processing time and throughput. Therefore, more research should be expected on task parallelism of ER.

3.4 Related Classifications and Surveys

In this section, we discuss related classifications surveys in terms of ER. As mentioned before, the research on ER in computer science has already a long history since the 1960s. There have been many surveys published, which address different aspects of the traditional ER and will be presented first. Afterward, the related work focuses on ER solutions to solve the challenges big data has brought.

Classifications and Surveys on Traditional ER:

The following surveys come down to broad aspects of the ER problem. Gu et al. [2003] described a record linkage system design and summarized common techniques used in each key system component, such as blocking, comparison, decision model,

etc. Besides, they provided new alternatives to implement these components and compared them to previous algorithms. Elmagarmid et al. [2007] gave a thorough analysis of approaches on duplicate record detection, which specifically includes techniques used to match records with a single attribute or multiple attributes, techniques for improving the efficiency and scalability of approximate duplicate detection algorithms, and a few commercial tools used in the industry. They also briefly discussed some open problems. Winkler [2006] touched on data preprocessing, pairwise comparison, and classification steps of ER and concluded existing research for each step. Besides, he also discussed approaches for automatic estimation of error rates, utilizing auxiliary information or creating functions and metrics to assist and improve matching, etc. Getoor and Machanavajjhala [2012, 2013] gave tutorials on ER, discussed existing solutions, current challenges, and open research problems for ER from various fields, including databases, machine learning, natural language processing, and information retrieval.

The following two articles focus on approaches to calculate similarities, although other aspects are also briefly covered. Brizan and Tansel [2006] first summarized distance matching techniques, then analyzed time complexity and the coverage of the constituent tuples for different applications including brute force, canopy, bucketing, hierarchical, data ming and mutual decision applications, and ways to apply them for different scenarios. Koudas et al. [2006] reviewed existing approximate matching predicates, which calculate the similarities between two data records. Besides, they concluded the pruning mechanisms to reduce the number of records pairs that need to be calculated and discussed possible ways to cluster data.

Christen [2012a] focused on indexing techniques used in ER and gave a detailed discussion of six techniques with a total of twelve variations of them. This included a theoretical analysis of their complexity and empirical evaluation of these techniques. Papadakis et al. [2016] focused also on blocking (also called indexing) techniques used in ER, which has the same focus as [Christen, 2012a]. They first classified 17 state-of-the-art blocking methods into lazy blocking, block-refinement, comparison-refinement, proactive blocking categories, and then empirically evaluated them on six popular real datasets and six established synthetic datasets.

Köpcke and Rahm [2010] compared and evaluated 11 proposed frameworks for ER. The comparison criteria adopted in this work can also be used to assess other frameworks. After the theoretical comparison of 11 frameworks, experimental evaluations were also provided to measure the effectiveness and efficiency of frameworks.

For ER's tightly related topic similarity join, there are also empirical comparisons and evaluations available. Jiang et al. [2014] gave a comprehensive survey on 14 string similarity joins. They first classified all algorithms into different categories based on the used filtering and verification techniques by different algorithms, then they provided experimental comparison results to compare the performance of different algorithms in different scenarios (including various input datasets with different properties, different similarity functions, etc). They ordered the compared algorithms in terms of efficiency under the aforementioned scenarios. Mann et al. [2016] conducted extensive experiments on seven set similarity join algorithms, which adopt a filter-verification approach. They pointed out that most of the approaches

put more effort into the filtering techniques to improve efficiency than the verification part, which cannot bring much efficiency improvement as expected. And more attention to the verification part should be paid.

Classifications and Surveys on ER for Big Data:

In recent years, along with the era of big data, there have been a large number of papers and articles that study ER for big data. Correspondingly several survey papers and tutorials exist in this area.

Christophides et al. [2019] provide an in-depth survey on ER techniques dealing with the challenges big data brings to it, especially focusing on blocking, filtering, matching, and clustering aspects of the batch, budget-agnostic ER. For blocking and filtering, they considered the non-learning and learning-based approaches for structured and semistructured data respectively. For the matching part, they discussed different matching methods based on schema-awareness, nature of comparisons, and algorithmic foundations. Besides, they also covered several special topics of ER including budget-aware ER, incremental ER, crowdsourcing ER, rule-based ER, and temporal ER. At last, open-source ER tools are also examined in their survey. El-Ghafar et al. [2017] provides a short overview of the approaches proposed to solve the volume challenge that big data brings to ER. They briefly discussed several important approaches, concluded the basic solutions to solve the volume challenge (partitioning data for parallel computation and applying MapReduce programming model), and proposed that research on load balancing strategies is required.

Vatsalan et al. [2018] gave an overview of big data issues in the special area of privacy-preserving ER. They proposed an analysis tool for analyzing, reviewing, and comparing existing approaches for PPRL and used their tool to conclude existing algorithms, focusing on the computational aspects, which are crucial for big data.

Gal [2014] gave a tutorial on ER, and he presented models and algorithms used for uncertainty in ER and exposed current challenges and future research directions. Another tutorial given by Papadakis and Palpanas [2016] focused on blocking-based ER, which includes not only some traditional blocking strategies but also the blocking techniques that have been developed for large-scale, heterogeneous data and web data.

For similarity join, Al-Badarneh [2019] concludes different types of join operations, which are implemented with the programming model MapReduce. Several important elements in the join operations including preprocessing, prefiltering, partitioning, replication, and load balancing, have been overviewed and discussed. Fier et al. [2018] experimentally compared ten MapReduce-based set similarity joins on 12 datasets with different characteristics. Surprisingly, their results show that all ten approaches failed to scale for at least one dataset, some of them even cannot work for small datasets. They analyzed the reasons to explain the poor performance of compared approaches and conducted more experiments to confirm their analysis. Based on those observations, they suggested some future research directions.

Based on the introduction of existing overviews and surveys on ER, we can see that, despite the importance of the research on parallel ER facing the big data volume

challenge and its prosperous stage, no survey focusing on parallel ER exist. To supply this gap, we conducted an SLR on parallel ER and reported our results. We hope this comprehensive overview and classification can reflect state-of-the-art research on parallel ER and also provide guidance on its future research.

3.5 Summary

In this chapter, we presented an overview and classification of publications on parallel ER based on three sets of criteria: general-aspect, effectiveness-based and efficiency-based criteria. General-aspect criteria include the specific operation types, number of input sources, and the input data type, which do not relate to any specific algorithms and indicate some fundamental considerations. Effectiveness-based criteria involve those criteria, whose purpose is to make ER more accurate, including data preprocessing, similarity function, match mode, and clustering. Efficiency-based criteria are the most important ones for parallel ER, which pursues reduced runtime, which include technologies used in blocking, data partitioning, load balancing, and redundancy handling. For those, we illustrated the most critical research questions: Which possible ways exist to efficiently partition data? As distributions of blocking keys may be uneven, which leads to data skew problems in parallel ER, how to balance the workload after the blocking step? Which specific measures can be taken into consideration to improve efficiency further? At last, we also discussed important open issues in the area of parallel ER.

4. Exploration on Performance Impacts of Different Implementations for Spark-Based Entity Resolution

This chapter shares material with the BDAS'18 paper "Exploring Spark-SQL-Based Entity Resolution Using the Persistence Capability" [Chen et al., 2018c] and the DEXA-BDMICS'18 paper "Performance Comparison of Three Spark-Based Implementations of Parallel Entity Resolution" [Chen et al., 2018a].

Among 4Vs of big data, data volume is the most intuitional character. The exponential growth of data raises the need for faster data analysis and processing speed. As introduced in the previous chapters, Entity Resolution (ER) as an intrinsic time-consuming task conforms to this requirement and parallel processing has been widely used to improve its efficiency and scalability. To implement parallel ER, there have been two main research directions so far: the first one is with parallel DBMSs, such as D-Swoosh [Benjelloun et al., 2007] and P-Swoosh [Kawai et al., 2006], while the other way is to employ a distributed computation framework to help with the implementation. The solution of using a parallel DBMS proposed more than two decades ago has some shortcomings for individuals or small and medium-sized enterprises with ER tasks. Parallel DBMSs are expensive [Stonebraker et al., 2010] and cannot operate in a heterogeneous environment and have very limited fault tolerance [Abouzeid et al., 2009]. As a result, users with limited budgets and strong fault tolerance requirements find it difficult to solve their ER tasks in parallel with a parallel DBMS. The second solution: employing a distributed computation framework has become very popular in recent years, since most frameworks are open-source, free to use, and also provide straightforward programming models to ease distributed programming. Based on our analysis of existing parallel ER approaches, the majority of approaches use the low-level APIs of Hadoop MapReduce or Apache Spark to

implement their parallel ER. They choose one implementation option randomly or based on their own preferences without considering the performance effects different implementation options may bring. Moreover, there has not been existing research to show the efficiency of different options, which can guide their choices. Under this situation, the chosen option may not provide the best efficiency for parallel ER.

To expose the performance impacts that different implementations with the big data processing frameworks may bring, in this chapter, we make a comparison study by exploring different implementation options withApache Spark as representatives of big data processing frameworks for implementing two common ER scenarios. Our main contributions are concluded as follows:

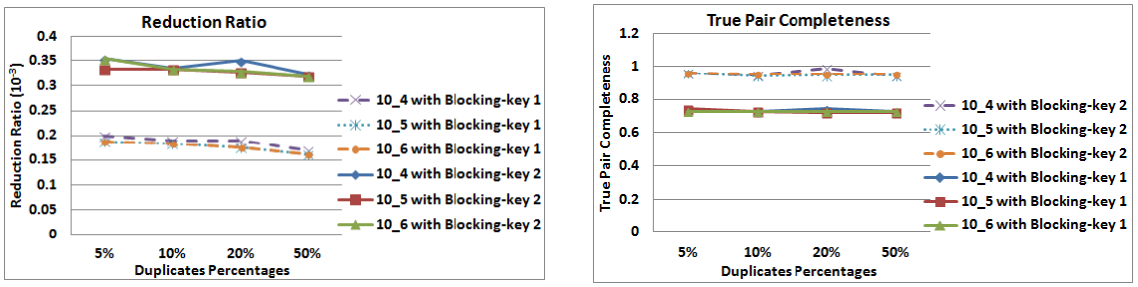
- We considered two scenarios of a common ER process and implemented them using three Spark APIs[1], respectively.
- We analyzed the ER process and designed different persistence options for Scenario 1, which may get benefits from persistence.
- We conducted experiments on both synthetic and real datasets and discuss the corresponding results.

The rest of this chapter is structured as follows: In Section 4.1, we introduce the two common ER scenarios we considered for evaluating the efficiency of different implementations and the corresponding Spark-based ER workflow. In Section 4.2, we present the details of different implementation options with Apache Spark. In Section 4.3, we introduce the experiments we conducted and discuss the corresponding results. In Section 4.4, we describe related work. At last, a summary of the whole chapter is provided in Section 4.5.

4.1 Entity Resolution Scenarios Used for Comparison

For our comparison study, we consider two ER scenarios: one is the common pair-wise ER process introduced in Chapter 2 with the evaluation step (Scenario 1) and one is the common pair-wise ER process introduced in Chapter 2 without the evaluation step (Scenario 2). The reason why we consider both scenarios is two-fold. On the one hand, two processes stand for two kinds of use cases. In some cases, people are concerned about only an approximate result of which records refer to the same entity without the need for knowing the exact precision and recall. Another case is without available ground truth, it is not possible to have the evaluation step. These two cases lead to an ER process without an evaluation step. For other cases, in which people need to know the result quality of ER and the ground truth is available, they lead to the ER process with an evaluation step. From a technical perspective, an ER process with an evaluation step involves the case of data reuse, while the process without an evaluation step does not involve data reuse. We want to know how data reuse affects the performance of different implementations.

[1]The source code and the datasets can be found at https://git.iti.cs.ovgu.de/Chen/entity-resolution-for-big-data.



(a) Reduction ratio.

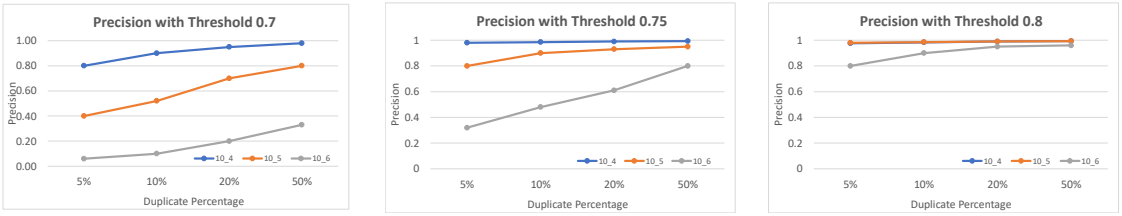
(b) Pairs completeness.

Figure 4.1: Blocking evaluation for blocking-key 1 and 2.

As mentioned above, for both scenarios, their first four steps are the same and follow a common ER process introduced in Chapter 2. Since the purpose of our research is to explore the efficiency of using different implementation options of big data processing frameworks to complete ER tasks, in general, we choose algorithms, strategies and corresponding parameters to conform to the tenet of high efficiency, but on the premise of satisfactory effectiveness. In the following, we introduce the first four steps of both scenarios in detail:

- *Data Preprocessing:* In this step, we reduce the distracting information by two common data cleaning methods: First, we remove characters or tokens that are useless or negative for the subsequent steps. Then, we fill those null values with "0".

- *Blocking:* Our blocking step applies standard blocking, which is straightforward and efficient, but able to achieve reasonable Reduction Ratio (RR) and Pairs Completeness (PC) by choosing a suitable blocking key. It is a trade-off to choose a more general or more specific blocking key based on the requirement for RR and PC. In our implementation for datasets with personal information, we experimentally tested two blocking keys. Because we define our blocking keys relating to names, we first apply a phonetic encoding function: Double-Metaphone on attributes "surname" and "given_name", which performs well not only on English names but also on European or Asian names. With this algorithm, letters that share a similar pronunciation are transformed to the same representation to handle common transcription mistakes that people might make when recording information based on what they hear from speakers. After "surname" and "given_name" are encoded, the first blocking key option is to concatenate the first two letters of the encodings followed by the first two numbers of attribute "postcode", while the second blocking key is only the concatenation of the first two letters of the encodings. As we can see from Figure 4.1, with the first more specific blocking key, the search space for pair-wise comparison was reduced by a factor of more than 5000 but around a quarter of true record pairs have been lost before the real comparison (PCs are around 75%). With the second more general blocking-key, the search space for pair-wise comparison is able to be reduced by a factor of 3000, which is almost two times more than the search space with the first blocking-key. However, in this case, a much higher PC can be achieved (for different datasets, PCs are



(a) Threshold 0.7. (b) Threshold 0.75. (c) Threshold 0.8.

Figure 4.2: Precision with different thresholds.

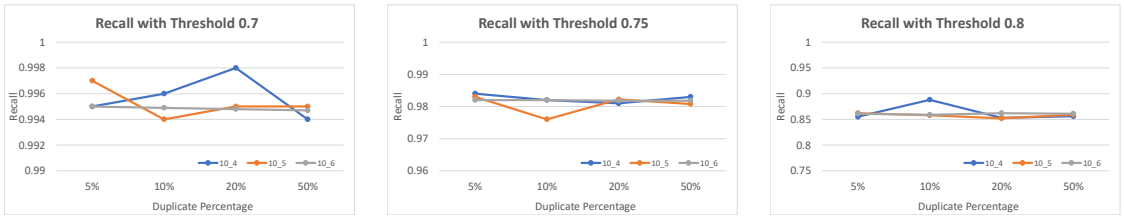
between 94.4% and 98.8%) and only a few true pairs have been lost before the pair-wise comparison step. Since PC of the first blocking key is not satisfactory, we choose the second one, which is a balanced solution for both RR and PC.

- *Pair-wise Comparison:* Almost all datasets have more than one attribute. Therefore, similarity functions need to be performed on each attribute to get a total comparison score afterward. Similarity functions should be chosen based on the attributes properties. In our case, we applied Jaro-Winkler distance for such attributes that are strings (including number strings), because Jaro-Winkler is proved to be a good and efficient edit-distance metric for short strings, such as for name matching [Cohen et al., 2003]. While for attributes that are numerical values, absolute difference functions are chosen because of their simplicity and understandability.

- *Classification:* In the classification phase, we classified each record pair to match or non-match based on the similarity scores of their attributes using a threshold-based method, which sums up all similarity scores to a total score and judges whether the score is higher than the threshold or not. If the score of a pair is higher than the threshold, this pair would be recognized as a matching pair. The difficulty in this step is how to choose a suitable threshold. Similar to the method of defining the blocking key, for our comparison study, the threshold is also determined by evaluating different thresholds. We have tested three different thresholds (0.7, 0.75, and 0.8) with different datasets and according to the ground-truth, we judged whether a threshold is good or not. As we can see from the results in Figure 4.2, for all datasets, with a higher threshold, precision is also improved. On the contrary, recall is reduced with a higher threshold (see Figure 4.3[2]). The reason behind is that a higher threshold leads to fewer matches and more non-matches. F-measure, as explained in Chapter 2, is a trade-off between precision and recall, and we take it as our standard to make the final decision. Figure 4.4 shows the F-measure results. For Datasets with 10^4 and 10^5, 0.75 is determined as the threshold. For Datasets with 10^6, 0.8 is determined as the threshold.

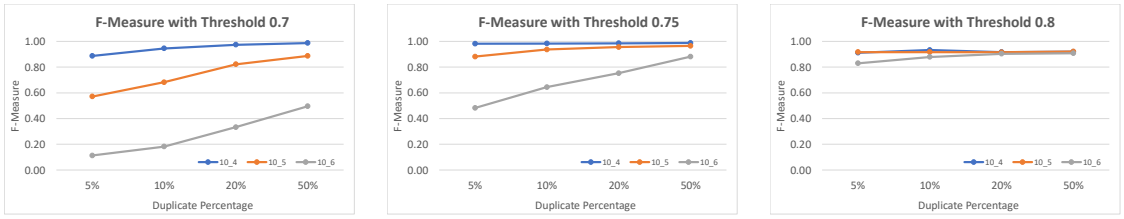
For Scenario 1, we evaluate the common metrics of effectiveness - precision, recall, and F-measure in its evaluation step. Next, we present how data is processed for a parallel ER process in Spark.

[2]The y-axes used in different sub-figures vary to clearly show the results.



(a) Threshold 0.7. (b) Threshold 0.75. (c) Threshold 0.8.

Figure 4.3: Recall with different thresholds.



(a) Threshold 0.7. (b) Threshold 0.75. (c) Threshold 0.8.

Figure 4.4: F-measure with different thresholds.

Parallel Data Processing of ER with Spark Figure 4.5 shows the steps for a Spark-based parallel ER and how the input data is transformed and processed to get the final ER results. The input datasets are located in HDFS or a connected database. As the starting step, the datasets are loaded into Spark and divided into a number of partitions. Data in the same partition will be processed by the same executor of Spark, and data in different partitions can be parallelly processed. Then with the preprocessing step, input data is transformed to *preprocessedData* by removing stop words and null values. Afterward, *preprocessedData* is transformed to *withKeyData*, because in the blocking step, the blocking key is generated and appended to *preprocessedData*. Subsequently, candidate pairs named *recordPairData* are generated through a join operation with the blocking key as the join attribute. For this step, data with the same blocking key from different partitions is placed together and relocated to a new partition, which causes shuffling. Next, similarities are calculated between attributes of all candidate pairs using preset similarity functions. Since the used classification is threshold-based, similarity scores of all attributes are summed up and stored as *totalScoreData*. With the classification step, each record pair is classified into a matching or non-matching pair by adding this information to *totalScoreData*, and *resultData* is obtained and saved back to the HDFS or the connected database.

For Scenario 1, extra steps for the evaluation step need to be done. For research purposes, synthetic datasets are often used due to the convenience to control their sizes and properties, and the ground truth used for synthetic may be contained by their IDs. For example, the ID of the synthetic dataset we used in our experiments is named *rec-ID_number-org* for an original record and *rec-ID_number-dup-duplicates_number* for a duplicate based on the original record. For the first group of records, the original record is called “rec-0000-org” and its duplicate records are “rec-0001-dup-0”, “rec-0001-dup-1”, etc. Therefore, for the evaluation step, we transform the *resultData* to *evaluationData* by removing the extra information and remaining

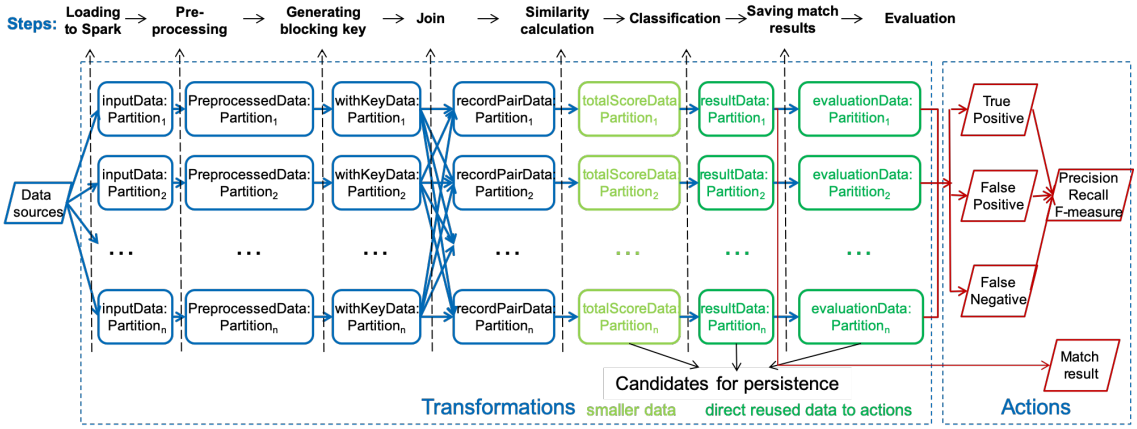


Figure 4.5: Spark-based ER.

the *ID_number* for all records. If the left two *ID_numbers* of a record pair are equal and their matching result is match or non-match, this record pair is a True Positive (TP) or a False Negative (FN) respectively. Otherwise, if the left two "ID_numbers" of a record pair differ from each other, this record pair is a True Negative (TN) for a non-matching pair result or a False Positive (FP) for a matching pair result. In this way, the number of TPs, FPs, and FNs are counted and used for the calculation of the evaluation results of precision, recall, and F-measure values. For the other common case, the ground truth is usually stored in an extra file with both IDs of a record pair and their true match or non-match fact. In this case, the ground truth file should be joined with the obtained *resultData*, and then according to the obtained match or non-match results and true match or non-match facts, all record pairs are classified into TPs, FPs, TNs, and FNs. Subsequently, similar to the former case, the number of TPs, FPs, and FNs are counted and used for evaluation.

4.2 Three Spark-Based Implementations for Parallel Entity Resolution

In this section, we introduce our three Spark-based implementations for the above-introduced ER process. The three implementations differ from each other due to the Java API they employ. As introduced in Chapter 2, there are three Java APIs in Apache Spark: the low-level RDD API in Spark core and two structured high-level APIs: the DataFrame API and the Dataset API. Therein, DataFrame and Dataset APIs have been unified to Dataset API in version 2.0 for ease of learning, in which DataFrame was considered as a special Dataset with a row type and renamed to Dataset<Row>. Nonetheless, for ease of expression in this thesis, we still use DataFrame API to stand for API of Dataset<Row>, while Dataset API means Dataset<T> API not including the special Dataset<Row> API. The difference between Dataset API (Dataset<T>) and DataFrame API (Dataset<Row>) is explained in the following. The return type of all SQL queries or SQL-like operations is Dataset<Row>, i.e., DataFrame, our DataFrame-based implementation is actually an SQL-based implementation. For our Dataset-based implementation, after we loaded the data as Dataset<T>, we do not use any API of Dataset<Row>, but other possible APIs for the general Dataset<T>.

4.2.1 Step-Wise Introduction of Implementation Details with Three APIs

Based on the data transformation steps described in Figure 4.5, we will introduce in the following how the steps are implemented with three Spark APIs.

Initializing Spark and Loading data into Spark (see Listing A.1 and Listing A.2): For the RDD-based implementation, *SparkContext* is first generated to allow the access to the Spark cluster with the help of the resource manager. Then the input dataset is loaded into Spark through *SparkContext*'s *textfile* method as an RDD abstract in the *String* type. For Dataset and DataFrame, instead of generating *SparkContext*, *SparkSession* is used for initializing Spark. Dataset loads data into Spark by the *textfile* method under *SparkSession*'s *read* method. DataFrame can directly load data through its general *load* method or specific methods designed such as for loading CSV, JSON, Parquet files.

Preprocessing (see Listing A.3): To implement the preprocessing step including the explained two common data cleaning steps, a *map* operation is used for RDD-based implementation. In this *map* operation, the RDD in the *String* type is transformed to our self-defined *record* type (*JavaRDD<Record> input*), which has all attributes and *blockingKey* fields and corresponding methods to enable the use of the required operations in the later steps. Unwanted characters or tokens are removed and null values are replaced using common methods for editing strings. As introduced in Section 2.3.1, to keep the advantages of RDD's type-safety and DataFrame's automatic optimizations, Dataset API is proposed. From the perspective of programming, for type-safety, the implementation with Dataset API is similar to the RDD implementation. Self-defined types are used to enable the use of the required operations. Due to the necessity of using *Encoder* to convert a JVM object of type T to the internal Spark SQL representation, corresponding types need to be defined for each object with different fields. In this way, more types need to be defined than using RDD API. For our ER process, specialized types for the original input record, record with a blocking key, the record pair data after join, data with separated scores, and calculated total score, result data, and evaluation data are defined. Similar to the RDD implementation, the transformation from a step to the next is implemented using *map* operations and procedures included in transformations are implemented using self-defined functions as well. These implementation methods are also the same for the following steps, and we will omit the same explanation for them. For DataFrame, the preprocessing step can be easily implemented with a *replace* method to remove useless characters or tokens, and a *fill* method from Class DataFrameNaFunctions of Spark SQL package to fill those null values with "0".

Blocking key generation (see Listing A.4): To generate the blocking key for records, a *map* operation is also used for RDD-based implementation, the blocking key is generated using self-defined functions and then is set to the *record* RDD (*JavaRDD <Record> withKeyData*). Regarding the blocking process in Spark SQL, we first created a temporary view based on the pre-processed data and then appended a new column named "blocking key" to the view using an SQL query.

Join for generating record pairs (see Listing A.5): For RDD-based implementation, to get candidate record pairs using the *join* method from PairRDDFunctions, a *mapToPair* operation transforms *withKeyData* from the type *JavaRDD<Record>* to *JavaPairRDD<String, Record>* by making the blocking key as the key and the record value as the value for the (key,value) pair. Then the join method performed as a self-join is applied to *JavaPairRDD<String, Record> withKeyPairData* and a filter operation is used to remove duplicate record pairs by only keeping the pairs, whose first record ID is smaller than the second ones. Because the blocking key as the key of the (key, value) pair is no more useful, a *map* operation transforms recordPairData from the type *JavaPairRDD<String, Tuple2<Record, Record>> recordPairData* back to *JavaRDD(Tuple2<Record, Record>)*. To perform the similarity calculations for the record pairs, a *TupleRecord* class is defined to contain two records attributes of all record pairs. For Dataset-based implementation, the *join* operation in the Dataset API is implemented with its method called *joinWith* and it does not allow self-join. For DataFrame, we performed a self-join on the current view using an SQL query as well, in which "blocking key" is used as the join attribute and record tuples join other tuples only if their attribute "id" value is bigger than those of other tuples as a join condition. This avoids generating record pairs repeatedly. In the meanwhile, we renamed all attribute names of the right record and deleted attribute "blocking key", which is no longer needed for our application. In this way, we can get all record pairs that need to be compared and each pair is represented as a tuple.

Similarity calculation and classification (see Listing A.6): For RDD-based implementation, *totalScoreData* with the type of *JavaRDD< TupleRecord >* is obtained continuing through a *map* operation with user-defined similarity calculation functions. At last, *resultData* is collected by transforming *totalScoreData* with a *map* operation to perform the threshold-based classification and saved to HDFS or a connected database with the *saveAsTextFile* method. For Dataset-based implementation, it is similar to RDD-based implementation. For saving the result, the function to save CSV files can be used. Regarding the DataFrame-based implementation, we can define required similarity functions through user-defined functions in Spark SQL to get their similarity scores on each attribute and get the total score by summing them up. To avoid transferring unnecessary information, the *selection* method is used to only keep the necessary data columns. Similar to this step, the classification step is also implemented by defining a classification function through user-defined functions. After matching or non-matching status is obtained, we appended this result as a new column in Spark SQL with the *withColumn* method and write back to HDFS or the connected database.

Evaluation (see Listing A.7): For RDD-based implementation, to evaluate the matching results, *recordsID* attributes of *resultData* is cleaned by removing extra characters and showing the ground truth and transformed to *evaluationData* within a *map* operation. Then TPs, FPs, FNs are counted with RDD's *count* method to calculate percison, recall, and F-measure results. Then the results are saved to HDFS or other connected databases with their corresponding methods, for example, to save results in HDFS, a file is first created and evaluation results are written to it with available methods of HDFS. The Dataset-based implementation is quite similar

to the RDD-based implementation. For the evaluation step with DataFrame, we first got the *evaluationData* by performing the user-defined function on *resultData*, and then TPs, FPs, and FNs are calculated with SQL *count* queries, and then, based on them we could calculate precision, recall and F-measure metrics and saved them also as a DataFrame named *evaluation* to HDFS by the built-in *write* method.

Comparing the three implementations, the DataFrame implementation is the most convenient one, which can use familiar SQL queries and many available built-in functions. In contrast, the Dataset implementation is the most complicated one due to the necessity of defining more types to use *Encoder* for type-safety.

4.2.2 Optimizations on Each Implementation

We optimize each implementation by tuning the following parameters: the level of parallelism and possible persistence options.

Parallelism Level

Choosing a suitable level of parallelism for RDD is crucial to reach good performance. The default level of parallelism for RDD is based on the input data size. For an ER task, for the case that the input data size is small, the data size for pair-wise comparison can be very large. Therefore, if the level of parallelism is determined by the input size, it cannot fulfill the parallelism requirement for steps after blocking. As a result, we have to tune the level of parallelism parameter to reach a good performance (After test experiments on different levels of parallelism are conducted, 320 is finally chosen for the RDD-based implementation). For DataFrame and Dataset APIs, the default level of parallelism is 200, which is proved to be a sufficient number in our experience and needs no specific tuning considerations.

Persistence

As described in Section 2.3.1, laziness is an important strategy in Spark to optimize performance and achieve fault tolerance. However, because of this property, the related data is re-evaluated through a repetitive transformation procedure, each time an action is triggered. This may lower the performance when no suitable persistence option is used. Therefore, persistence needs to be seriously considered for a process with data reuse. As introduced above, only Scenario 1 involves data reuse, therefore, persistence options are only necessarily considered for Scenario 1.

There are two aspects that need to be considered to find the best persistence option. The first one is which persistence level is used for persisting the data, the other one is which data to persist can improve efficiency the most. For the first aspect, the persistence level taken is MEMORY_ONLY for all experiments to achieve the best performance in Spark. For the other aspect, we analyze the ER process of Scenario 1 to find candidate data to persist, which are then tested experimentally to find the best persistence option. Next, we will introduce the process to find candidate data for persistence. Straightforwardly, the data that is involved in multiple actions should be persisted so that the runtime can be reduced by avoiding the repetitive transformation procedure. Figure 4.5 shows transformations and actions in our ER process. As we can see from it, *resultData* and *resultCleanedData* are directly reused data by multiple actions including saving match results, and several calculations for

Table 4.1: Datasets used in experiments.

Datasets \ property	ID	Name	Input Size
Synthetic Datasets	1.1	$5 * 10^5 + 5\%$	57.3M
	1.2	$5 * 10^5 + 50\%$	79.9M
	1.3	$10^6 + 5\%$	114M
	1.4	$10^6 + 50\%$	160M
Real Datasets	2	Facebook	82.7M

evaluation. Therefore, they are considered first as candidate data for persistence. Thus, how much benefits can be achieved through persistence options depends upon two factors: one is how much we can save by avoiding repetitive transformation, and second, how much persistence overhead it costs. If we want to get the most benefit, we need to avoid as long as possible the repetitive transformation procedure, which means persisting direct reused data by actions, and meanwhile, reducing persistence overhead as much as possible, which means persisting data which is the smallest. However, data that fulfills both conditions may not exist and it is also not possible to analyze which one is more important for deciding the final benefit. In our case, *resultData* is the directly reused data, but it is much larger than *totalScoreData* and the transformation from *totalScoreData* to *resultData* is only dropping one column without any other complicated calculation. Therefore, we take *totalScoreData* together with *resultData* and *resultCleanedData* into account as candidate data for persistence. In conclusion, we have three candidate data: *totalScoreData*, *resultData*, *resultCleanedData*. Based on them we can have different persistence options, which will be evaluated in our experiments to figure out the best persistence option.

4.3 Evaluation

In this section, we present the experiments we conducted for the comparison study. In Section 4.3.1, we introduce our experimental setting including datasets and the Spark cluster we used. In Section 4.3.2, we show and discuss our experiment results for Scenario 1, which includes the experiment results of evaluating different persistence options, runtime, and speed-up comparison of different Spark implementations. In Section 4.3.3, we show and discuss our experiment results for Scenario 2, which compares the runtime of different Spark implementations.

4.3.1 Experimental Setting

In this subsection, we introduce our experimental setting. It relates to two aspects. On the one hand, we describe the datasets that we used to evaluate our application; on the other hand, we demonstrate our Spark YARN cluster based on the Hortonworks Data Platform (HDP).

Datasets for Experiments. The datasets used are shown in Table 4.1, which include synthetic and real datasets. By using synthetic data, their ground truth is easy to get and their properties, such as sizes or duplicate percentages of datasets, can be controlled. The synthetic datasets are generated using GeCo introduced

in Section 2.3.2 and are stored in CSV files. The datasets are named based on their sizes and duplicate percentages. For example, Dataset $10^6 + 5\%$: the first part before the plus sign means the number of original records that a dataset contains, i.e., 10^6 means there are 1 million original records. The second part represents the duplicate percentage based on the original records, i.e., 5% means the number of duplicates is 5% of the number of original records and is inserted into the dataset. Because of privacy reasons, it is very hard to find real datasets that contain personal information. The real data we have for the experiments are parsed public data from Facebook [Bowes, 2010], which are people's names including duplicates. There is no ground truth for this dataset, so we conducted experiments with real datasets to evaluate the efficiency and speed-up of Scenario 1 without the evaluation step. Our real dataset, which is shown in the last row of Table 4.1, includes one million records used for evaluating efficiency.

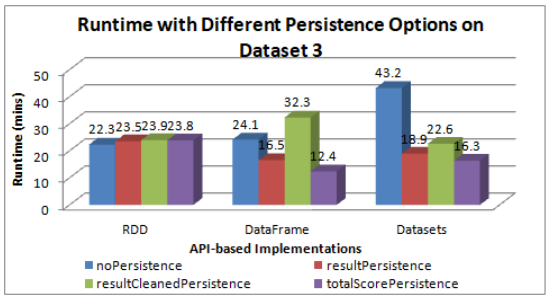
Spark YARN Cluster. We have implemented our Spark-SQL-based ER in Java and conducted all experiments in our Spark YARN cluster with the version Spark 2.0. We use HDP to deploy our Spark YARN cluster, which is an open-source Apache Hadoop distribution based on a centralized architecture (YARN) [Hortonworks, 2020]. The cluster has ten Spark clients, which includes one node to access the cluster, two nodes as HDFS NameNodes (one active and one standby), and seven executor nodes as HDFS DataNodes, which can also be used to run Spark applications. Each of them runs on virtual machines, whose hyper-visor is VMWare ESXi. In the following, we introduce hosts' hardware for seven executor nodes. Each of them has four CPU cores, 16GB RAM (6GB is available for executor nodes to run Spark applications), and 150GB hard disk. However, our cluster is heterogeneous, since four of nodes are with four Intel Xeon E5-2650 @2.00GHz cores, two nodes are with four Intel Xeon E5-2650v2 @2.60GHz cores, the last nodes are with four Intel Xeon E5520 @2.27GHz cores. All hosts are connected through a 10GBit Ethernet with a star topology. We submitted our application with jar files to our Spark Yarn cluster and conducted a series of experiments to evaluate the runtime, speedup of our baseline implementation, and our implementation with persistence. We run each experiment five times and after dropping the highest and lowest result, our final result was obtained by averaging the remaining three results.

4.3.2 Scenario 1: with an Evaluation Step

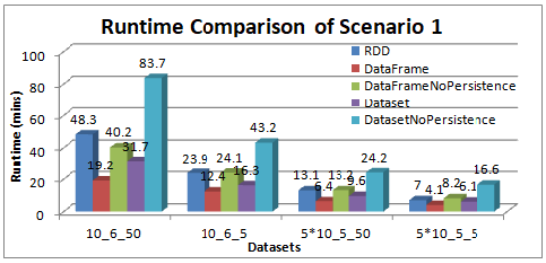
This section shows the comparison results for Scenario 1. The first group of experiments tests different persistence options for all three implementations to find their corresponding best persistence option. Then the second group of experiments compares the runtime of three implementations on different datasets. At last, the speed-up of all three implementations is compared.

Evaluation: Persistence

Experiment Design. According to the analysis in Section 4.2.2, we have the following persistence options: noPersistence, resultDataPersistence, resultCleanedDataPersistence and totalScoreDataPersistence. For the "noPersistence" option, we do not persist data. The other three options are persisting these three candidate data respectively. We ran these implementations with different persistence options on different datasets with our 7-executors-cluster and recorded their runtime respectively.



(a) Evaluation of four different persistence options.



(b) Runtime comparison.

Figure 4.6: Persistence evaluation and runtime comparison of Scenario 1.

Persistence level "memory_only" is chosen to provide the most efficient persistence, because our cluster has sufficient memory to store candidate data.

Results and Discussion. Figure 4.6a shows an overview of runtime on three implementations with four different persistence options on dataset 3 for Scenario 1. The results on other datasets show a similar trend. Surprisingly, there is no obvious performance difference with those four persistence options for the RDD-based implementation. Even with persisting the relevant data, Spark runs the RDD-based implementation in the same way, which leads to similar runtime. Because persisting data for the RDD-based implementation cannot avoid overhead, for the RDD-based implementation, we did not use any persistence for it. In contrast, DataFrame- and Dataset-based implementations can benefit from persistence. As we can see from Figure 4.6a, by persisting result data, an improvement of a factor up to 1.5 and 2.3 for DataFrame-based and Dataset-based implementation can be achieved, respectively. By persisting *totalScoreData* instead of the straightforward choice: *resultData*, efficiency improvement can reach a factor up to 2x and 2.7x for them. We conclude that the straightforward choice to persist is not always the best, when the data that is used for multiple actions is quite large. Instead of directly persisting it, we can judge whether it is possible to persist its former data that is much smaller than it and when the computation between them is not time-consuming and their data sizes differ much, it is normally beneficial to persist the former one instead. Therefore, we take persisting *totalScoreData* as the best persistence option for DataFrame- and Dataset-based implementations. Since the RDD-based implementation cannot benefit from persistence options, for runtime comparison we also took DataFrame- and Dataset-based implementations without any persistence into consideration.

Runtime comparison

Experiment Design: Based on the above discussion on persistence options, for Scenario 1, we have five different cases to compare their runtime: RDD-based without persistence, DataFrame-based without persistence, DataFrame-based with the best persistence, Dataset-based without persistence, Dataset-based with the best persistence. Each case is submitted to our Spark cluster with seven available nodes on different datasets and their corresponding runtime is recorded.

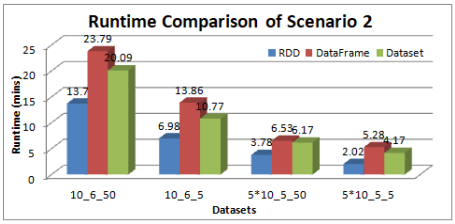


Figure 4.7: Runtime comparison of Scenario 2.

Results and Discussion. Figure 4.6b shows the comparison result. As we can see from it, the DataFrame-based implementation with the best persistence is the most efficient one, which is up to 2.5x and 1.6x faster than the RDD-based implementation and the Dataset-based implementation, respectively. However, if without persistence, the RDD-based implementation has similar speed as the DataFrame-based implementation, and sometimes is even slightly faster than the DataFrame-based implementation, but both of them are much faster than the Dataset-based implementation. In conclusion, by using a proper persistence option, the DataFrame-based and the Dataset-based implementation are able to outperform the RDD-based implementation due to the obtained benefits from persistence. The RDD-based implementation did not change its running plan with the persistence options and cannot get benefits from persistence in Scenario 1.

4.3.3 Scenario 2: without an Evaluation Step

For the second scenario, our ER process does not include the evaluation step and we removed this step from Scenario 1 and kept the rest of the steps as Scenario 2. Since there is no data reuse in Scenario 2, it is not necessary to have any persistence options. We submitted three implementations to our 7-nodes-cluster on the four datasets introduced in Table 4.1. Figure 4.7 shows the results. For all datasets, it shows a consistent result: among three implementations, the RDD-based implementation is the fastest, up to 2.6x and 2.1x faster than the DataFrame-based and the Dataset-based implementation, respectively. The DataFrame-based implementation is the slowest, even slower than the Dataset-based implementation, which is the opposite side to Scenario 1 when we consider the best persistence case for all of them. In conclusion, expected physical and logical optimizations do not help the DataFrame-based and the Dataset-based implementations much. The RDD-based implementation shows the best efficiency for running a general ER process without the evaluation step (without data reuse).

4.4 Related Work

In this section, we present the related work of our comparison study. Along with the growing popularity of Apache Spark in recent years, there have been several implementations based on Apache Spark for parallel ER.

Pita et al. [2015] implemented a Spark-RDD-based approach for probabilistic ER of healthcare data. Particularly, in their evaluation part, they compare the Spark-based implementation with an OpenMP-based implementation. Their results show

that an OpenMP-based implementation is much faster than Spark-based ER. However, Spark-based ER has the scalability and fault tolerance advantages compared to OpenMP-based ER. Mestre et al. [2017] propose S-DCS++, a Spark-RDD-based ER with an adaptive-window Sorted Neighborhood blocking method and a load balancing strategy. In their evaluation, they also compared a Spark-based implementation to a MapReduce-based implementation. Their results show better efficiency of Spark-based implementation as well.

Dou et al. [2019] extended a density-based blocking strategy DUB to the distributed environment. Because the naive distributed solution NDUB suffers from frequent communications between nodes, which leads to low efficiency, they proposed DDUB, which randomly splits the input dataset into several nodes and approximately estimates the global density based on local densities instead of getting accurate global density through communications between nodes. They implemented NDUB and DDUB with both Hadoop MapReduce and Spark and evaluated them. The results show that MapReduce-based NDUB works much worse than the other three implementations and their proposed DDUB approach performs well for both Hadoop MapReduce and Spark, although Spark-based implementation achieves slightly higher efficiency.

There are also several similarity join approaches implemented with Spark. Chen et al. [2015] focused on the Top-k similarity join problem, which is akin to ER, and implement a Spark-RDD-based algorithm for massive multidimensional data, focusing on a more efficient distance function using locality sensitive hashing (LSH) and determining the top-k closest pairs. For their evaluation, they also explored using persistence to improve the efficiency of Spark-based ER, and compared the runtime of their Spark-based implementation with and without persistence with the Hadoop Reduce based implementation. Their results show that Spark-based ER can benefit from persistence and provides better results than MapReduce-based ER even without persistence acceleration. Wang and Karimi [2016] focus on using a k nearest neighbors (kNN) algorithm for the classification step with Spark and propose a method to minimize the cross-cluster kNN search. Some pruning approaches have been proposed to improve efficiency. Rong et al. [2019] proposed their dimension reduction and vertical partitioning algorithm for Spark-based similarity join, which can provide balanced loads for different nodes. They also implemented their algorithm with both Hadoop MapReduce and Spark and got a similar conclusion to the aforementioned approaches that Spark-based implementation provides better efficiency.

Compared to the above-introduced Spark-based ER or similarity join approaches, which focus on proposing approaches for ER and compare efficiency between Hadoop MapReduce based and Spark-based implementations, our comparison study considers two general ER scenarios and implemented them with three Spark APIs. The purpose of our study is to compare the different implementations of parallel ER with both low-level and high-level APIs available in big data processing frameworks and particularly conclude their performance in terms of efficiency under different scenarios. As three APIs including both low-level and high-level APIs are available in Spark, we take them as an example for the comparison study and its persistence option is also discussed to achieve the best performance for Spark-based ER.

4.5 Summary

In this chapter, we compare the performance of using three APIs: RDD-based API, DataFrame-based API, and Dataset-based API in Spark for two scenarios of a general ER process. For Scenario 1, RDD-based implementation runs faster than the other two implementations without consideration of any persistence option for them. However, the DataFrame- and Dataset-based implementations benefit from persistence, but the RDD-based implementation does not. Therefore, with the best persistence option for DataFrame- and Dataset-based implementations, they are able to outperform the RDD-based implementation. For Scenario 2, the RDD-based implementation is the fastest, which conforms to the case in Scenario 1 without consideration of any persistence option. In conclusion, our work shows that implementations based on different APIs differ in their efficiency and there is no best solution for both scenarios. In the adoption of Spark Dataset or DataFrame API for ER tasks, developers are susceptible to impact their runtime by using incorrectly the persistence options of Spark. We believe that there is a need either to propose a "best practices" approach for nudging developers into using the best solution or to add these carefully adopted persistence optimizations into the Catalyst optimizer, by using statistics and cost estimations similar to relational databases. Except for the above-mentioned observations, we also realized that with the default load balancing strategy of a big data processing framework the reducer workload could not be balanced facing skewed block, which heavily increased the runtime of a parallel ER process and lowered its speed-up. To solve this problem, in the next section, we will propose a scalable and robust skew handling strategy to balance the reducer workload.

5. Analysis and Comparison of Block-Splitting-Based Load Balancing Strategies for Parallel Entity Resolution

This chapter shares material with the iiWAS'20 paper "Analysis and Comparison of Block-Splitting-Based Load Balancing Strategies for Parallel Entity Resolution" [Chen et al., 2020].

As introduced in the previous chapters, in the area of parallel Entity Resolution (ER), big data processing frameworks, such as Hadoop MapReduce or Apache Spark, are often used to implement parallel ER [Chen et al., 2018b]. Those frameworks conform to the "map" and "reduce" programming model. During the map phase, the input data is transformed by generally adding the blocking key to each record, then the intermediate map output data is repartitioned using their default strategies hash or range partitioning to the reducers, in which the real record comparisons happen. By range partition, given n blocks and m reducers, blocks are first sorted based on their keys, then they are divided into m ranges based on the sampling method and each range of blocks is assigned to one reducer [Xin et al., 2014]. By hash partition, given n blocks and r reducers, a hash function is applied to all blocking keys to assign them to r reducers [Zaharia et al., 2012].

One major drawback from these strategies for parallel ER, is that they only balance the load when the keys are evenly distributed. For parallel ER with blocking, the sizes of each block usually show a great variation due to the different number of records with different blocking keys. Then all records with the same blocking key are assigned as one partition to a reducer, which leads to an uneven distribution of the whole workload to available reducers. Furthermore, the pair-wise similarity comparison, which is performed on the reducers, amplifies the impact of the skew. Consequently, the reducer with overpopulated blocks will dominate the runtime of the whole ER process, which will seriously degrade the performance of parallel

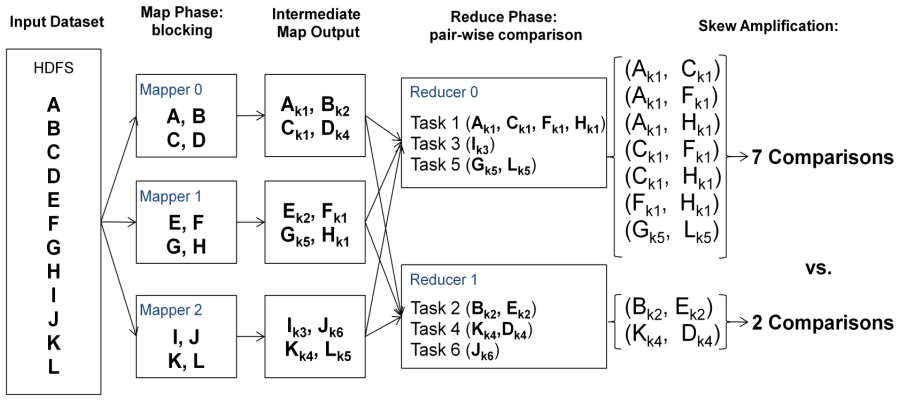


Figure 5.1: An example of load imbalance with hash partition.

computation. Figure 5.1 shows an example of this problem. The input are 12 records, labeled A to L, and four of them (A, C, F, H) are with the blocking key k_1, records B and E, K and D, G and L are with the blocking key k_2, k_4 and k_5 respectively, the remaining two records I and J are with the blocking key k_3 and k_6 respectively. All of them are sorted by their blocking keys after the map phase and grouped to tasks. One task containing records with the same blocking key cannot be divided and should be assigned as a whole to one reducer. As an example of hash partition, tasks 1, 3, 5 are hashed to reducer 0, and tasks 2, 4, 6 are assigned to reducer 1. In this way, reducer 0 holds 7 records, which leads to 7 comparisons, and reducer 1 holds 5 records, which leads to only 2 comparisons. With the pair-wise comparison step, the skew on blocking keys is further amplified. During our comparison study in Chapter 4, we employed the default load balancing strategy and we observed an overlarge distance between our implemented speed-up and the ideal speed-up exists, which is mainly caused by the unbalanced load among different reducers. Therefore, to achieve the best speed-up for the ER process by combining the use of blocking and parallel computation, it is of great importance to detect possible uneven workload distributions and develop skew handling strategies to improve efficiency and scalability.

So far, there have been different approaches proposed to solve this load imbalance problem, which can be divided into two research directions: block-splitting-based and pair-based. Block-splitting-based solutions are straightforward and effective [Chen et al., 2018b]; they remove bottleneck blocks by splitting them into several sub-blocks. The other direction: the pair-based load balancing stategy permutes all record pairs and then evenly assign them to reducers. However, one typical pair-based strategy PairRange [Kolb et al., 2012b] increases its map output size linearly with increasing reduce tasks, and often has longer runtime than BlockSplit. Therefore, in this chapter, we focused on the first direction and study the problem of providing a balanced reducer load for parallel ER using block-splitting-based methods. We conclude our contributions as follows:

- We first analyse the state-of-the-art of block-splitting-based load balancing strategies and point out their advantages and disadvantages.

- Afterward, based on the observations, we propose two strategies Two-Level Split (TLP) and Block-Oriented Slicer (BOS), which overcome the identified drawbacks.
- We implement[1] and evaluate our proposed strategies, and compare them with the state-of-the-art approaches: BlockSplit and BlockSlicer, and the default load balancing strategy in Apache Spark. Based on our evaluation results, we report the winners under different circumstances and conclude the important factors that affect the performance of a block-splitting-based load balancing approach.

The remainder of this chapter is organized as follows: In Section 5.1, we describe the common workflow of a block-splitting-based load balancing strategy. Then in Section 5.2, we first analyze two state-of-the-art strategies BlockSplit and BlockSlicer by extracting their advantages and disadvantages. Subsequently, we propose two new strategies Two-Level Split (TLS) and Block-Oriented Slicer (BOS) to overcome the identified drawbacks. Afterward, we evaluate the four strategies comprehensively, along with the default strategy of Apache Spark in Section 5.3. In Section 5.4, we describe related work and we conclude our work in Section 5.5.

5.1 The Common Workflow of a Block-Splitting-Based Load Balancing Strategy

As explained above, skewed blocks are the primary cause for load imbalance in parallel ER. Our research studies the strategies that split the overpopulated blocks into multiple sub-blocks to eliminate the bottleneck, whose fundamental research goal is to provide a balanced reducer workload. Despite the overhead to split the blocks, it is optimal if all blocks have the same size, which means no skew between blocks. Under this assumption, an average block size should be calculated. Based on the average size, all overpopulated blocks are first split to sub-blocks. The sizes of all sub-blocks are equal to the average block size except the last sub-block. Then the blocks smaller than the average block size should be combined. However, the overhead to complete the above steps will be large and severely reduce efficiency. Therefore, a feasible method is to solve the reduce imbalance problem caused by skewed blocks within two steps. First, the splitting algorithm is only used to reduce the skew degree. Afterward, a proper reducer assignment strategy can help to reach the final goal - a balanced reducer workload.

Figure 5.2 shows the workflow of a block-splitting-based load balancing strategy. Besides these two main steps mentioned above, before splitting the overpopulated blocks, the block distribution information should be gathered and a threshold to distinguish regular and overpopulated blocks is calculated. Then the overpopulated blocks are split. The details of four strategies will be shown in the next section. Afterwards, the record pairs need to be compared are generated based on their composite keys. At last, the similarities between each record pair are calculated, which are used to classify the pairs into matches and non-matches, and the ER results are saved.

[1]The source code and the datasets can be found at https://git.iti.cs.ovgu.de/Chen/entity-resolution-for-big-data.

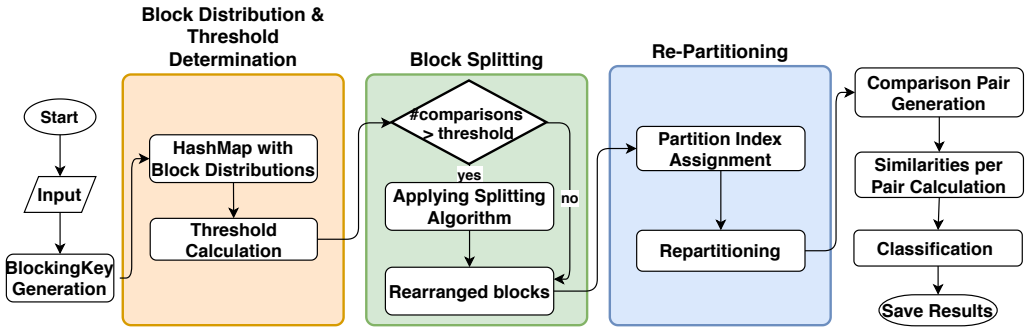


Figure 5.2: Workflow of a Block-Splitting-Based Load Balancing Strategy for ER

5.2 Block-Splitting-Based Load Balancing Strategies

So far, there have been the following strategies proposed for the block-splitting-based load balancing strategy [Gomes Mestre and Pires, 2013; Kolb et al., 2012b; Yan et al., 2013a]. The research in [Yan et al., 2013a] is mainly to improve scalability by using an approximate sketch data structure for gathing block distributions, which is at the expense of larger imbalanced reducer load. This research goal does not conform to the main research goal of a load balancing strategy, which concern much about a balanced reducer. For the worst case, when the memory is not sufficient to hold the block distribution information, it is possible to store them in a distributed storage as proposed in [Kolb et al., 2012b]. Therefore, in this section, we focus on the left two state-of-the-art approaches BlockSplit and BlockSlicer. The existing comparison between them is quite limited, and only speed-up experiments with a fixed setting of map and reducers are represented in [Gomes Mestre and Pires, 2013]. To fill the gaps, in this section, after an introduction to the approach to get the block distribution statistics and determine a threshold to identify overpopulated blocks, we analyze the advantages and disadvantages of these two methods.

5.2.1 Block Distribution Collection and Threshold Determination

Block Distribution Collection: To distinguish regular and overpopulated blocks, the block distribution statistics need to be first collected. For strategies BlockSplit and BlockSlicer, they use a Block Distribution Matrix (BDM) to collect required information, which contains the number of comparisons of a blocking key on each mapper. For our comparison analysis for block-splitting-based load balancing strategies, we also uniformly use the BDM. It should be noticed that, with a greedy reducer assignment strategy, only BlockSplit requires a seperate information for the number of comparisons of a blocking key on each mapper. Other strategies only need to know the total number of comparisons of each blocking key. For implementation, a HashMap can be used, which contains triples of the form (blockingKey, mapper index, #records) [Kolb et al., 2012b]. The number of comparisons in one block can be calculated with the following equation based on the number of records:

$$\#comparisons = \frac{\#records * (\#records - 1)}{2} \tag{5.1}$$

Thresholds Determination: After the block distribution statistics have been collected, a threshold needs to be determinated as the input for the splitting algorithms. With a larger threshold value, fewer blocks are considered as overpopulated blocks and need to be split. In order to keep the overhead for splitting blocks small, the average reducer load should be used as the threshold. To calculate it, the total number of comparisons needs to be calculated. Then the average reducer load can be calculated by dividing the total number of comparisons by r reducers (given n blocks in total):

$$\theta_1 = \left\lceil \frac{\sum_{1st\ block}^{nth\ block} \#comparisons}{r} \right\rceil \tag{5.2}$$

Because the splitting of blocks are realized by assigning new composite keys to each sub-block, for those blocks, whose number of comparisons is smaller than the threshold value, i.e., regular blocks, their keys remain the same and no extra handling is required. The compositeKey "blockingKey" or "blockingKey, markForRegular" is assigned to them, which is used to indicate that they are regular blocks, in which each record is compared to any other records of the same block. The generation of the composite keys for large and huge blocks will be introduced next.

5.2.2 Overpopulated Block Handling

In this section, we focus on the steps to handle overpopulated blocks with the two aforementioned state-of-the-art methods BlockSplit and BlockSlicer. We provide first a concluded idea on how they work, and then expose their advantages and disadvantages. For detailed introductions we refer to the original papers by Kolb et al. [2012b] and Gomes Mestre and Pires [2013].

In order to remove the bottleneck blocks, i.e., overpopulated blocks, a block-splitting-based approach rearranges the contained comparisons to several sub-blocks. The sub-blocks are defined by designing composite keys based on the original blocking key for each record or each block.

BlockSplit: BlockSplit is the pioneer of the block-splitting-based load balancing strategy for parallel ER. Based on the map partitions, the blocks are first natrually split into m sub-blocks (m is the number of mappers). Subsequently, in order to maintain all candidate pairs for comparisons, records in one mapper need to be compared to all records in its own mapper or all other mappers. To realize and follow this principle, each record in an overpopulated block is replicated m times. Therein, the original record is assigned a composite key "*blockingKey, 0*", which is resonsible for the comparisons between records from the same partition. The other replicates are assigned composite Keys "*blockingKey, i, j*" (i and j are their mapper indexes), which are responsible for the comparisons between records from the ith and jth mapper. Then it is simple to get the candidate pairs based on the composite key. If a pair of records is from the same mapper, i.e., i is equal to j, then all records with the same composite key need to compare with all other records. Otherwise, records only need to compare with records from the same input partition.

As we can see from the BlockSplit strategy, the performance of BlockSplitthe heavily depends on the input map partitions. The calculated hreshold is only used to

distinguish regular and overpopulated blocks, not used to provide an upper bound for blocks. It is still possible to generate larger blocks than the threshold if too many records are partitioned into one mapper. The obtained sub-blocks, which are from the overpopulated blocks, are of different sizes based on the mappers as well. Besides, along with the increasing number of mappers, the records that need to be replicated are also significantly increased. Correspondingly, its efficiency can be expected to be reduced due to the increasing overhead.

BlockSlicer: In order to reduce the number of required replications for records in BlockSplit, BlockSlicer splits the overpopulated blocks in a different manner. It groups all records belonging to one block together, ignoring their mappers. Then the threshold is not only used to distinguish regular and overpopulated blocks, but also a standard to split the overpopulated blocks (the comparisons of the obtained sub-blocks may exceed the upper bound a bit, because its algorithm returns the number of records, with whose comparison the total comparisons of the sub-block is closest to the threshold). The basic idea is to consider each block as a list containing all its records, the number of comparisons in one block is calculated by summing up all necessary comparisons of all records (necessary comparisons mean the comparison between the current record and all its latter records). Then the sub-blocks, except the last one, should have similar number of comparisons as the threshold shows. After one iteration round, only the records that have not completed their comparisons are replicated to the next sub-block and the records that are replicated are marked with a star. To get the candidate pairs, for all records in a blocks, as long as the record does not have a star mark, it needs to compare with all its following records. If a record has a star mark, then it will not compare to its following records. In this way, the comparisons between two records both with star marks are not allowed to avoid redundant comparisons.

As we can see from the BlockSlicer strategy, it leaves many blocks after the splitting, whose number of comparisons is nearly equal to the threshold. This maybe not memory-efficient. In contrast, the default hash or range partitioning, which calculates the similarities between a record pair independently and does not need to put all record pairs together, will seldom encounter memory problems. However, with block-splitting-based load balancing approaches, the records from a same sub-block need to be grouped together to generate a list for candidate pairs to get the final ER matching result. With more blocks that have a similar size to the threshold for BlockSlicer, the returned list is more likely to occupy too much memory, and the GC time will become longer than BlockSplit, it may even fail to complete the matching process. While an overpopulated block is splitted often more times in BlockSplit, which suffers less from this problem.

For both BlockSplit and BlockSlicer, the composite key is assigned to each record. For the reducer assignment step, all records with the same composite key need to be grouped together, which may raise an overhead and increase the time for repartitioning.

Based on the above in-depth analysis of BlockSplit and BlockSlicer, we propose two other block-splitting-based strategies in this section. They are designed to overcome the two identified drawbacks of BlockSplit and BlockSlicer: performance

dependency on input mappers and record-wise composite-key assignment. Next, we will introduce these two strategies in detail.

Two-Level Split: To make the solution independent of the mapper, all records belonging to a same original block should be put together into one list and the number of sub-blocks that it is split into should not be related to the number of mappers. Although there is no relationship between our splitting algorithm and the mappers, the process to split the overpopulated blocks can be considered as two levels. This is similar to the idea in BlockSplit, which first partitions the overpopulated blocks into several sub-blocks, and then each sub-block is split to several parts to maintain a correct number of comparisons by considering both intra-sub-block and inter-sub-block comparisons. In our strategy, each overpopulated block is firstly divided into n first-level sub-blocks (n is calculated by dividing its number of comparisons by the average reducer load). This conforms to the straightforward ideas that the bigger the block is, the more pieces it should be split into, which is also similar to the splitting idea in BlockSlicer. The threshold is the upper bound for the re-arranged blocks as well to guarantee the balanced reducer load. For each first-level sub-block, we assign a mark that is equal to their block index to its record. To keep a same matching result, each record in one first-level sub-block needs to compare to all other records from the overpopulated block, no matter whether they belong to a same sub-block or not. Therefore, the composite key, which is composed of the blocking key, its own first-level sub-block index, and another first-level sub-block index, is used to generate the second-level sub-blocks. In this way, the composite key can be considered to be assigned to each second-level sub-block. Given a second-level sub-block with the new assigned composite key, if the second part is equal to the third part of the composite key, which means it is for comparisons within one first-level sub-block, its records compare to each other. Otherwise, the second-level sub-block is for comparisons between different first-level sub-block, a record only compares to another record from different first-level sub-block.

Block-Oriented Slicer: As explained in the above, the BlockSlicer strategy assigns the composite key to each record. In order to further reduce the overhead, based on the BlockSlicer's idea: considering each block as a list containing all its records, the number of comparisons in one block is calculated by summing up all necessary comparisons of all record. In order to be able to assign the composite key directly to each sub-block, in our strategy, we do not mark the replicated block with a star. Instead, for each sub-block, we add the information of the number of records that compare themsevles to all its latter records to the composite key for each sub-block. Therefore, the composite key for each sub-block is composed of the blocking key, the number of records with comparisons, and the iteration index. The first n records (n is second element of the composite key) are compared to all its following records.

Splitting Example: For ease of understanding, we give an example to describe how the comparison rearrangement proceeds for the four aforementioned strategies to show their differences (see Figure 5.3). Given a block with 8 records and a threshold 10, the number of comparisons of the original block is 28, which is much more than 10. Therefore, we need to rearrange all 28 comparisons to several sub-blocks.

As we can see from Figure 5.3, for BlockSplit and TLS, their splitting process can be divided into two levels. The first-level sub-blocks are obtained by simply splitting the

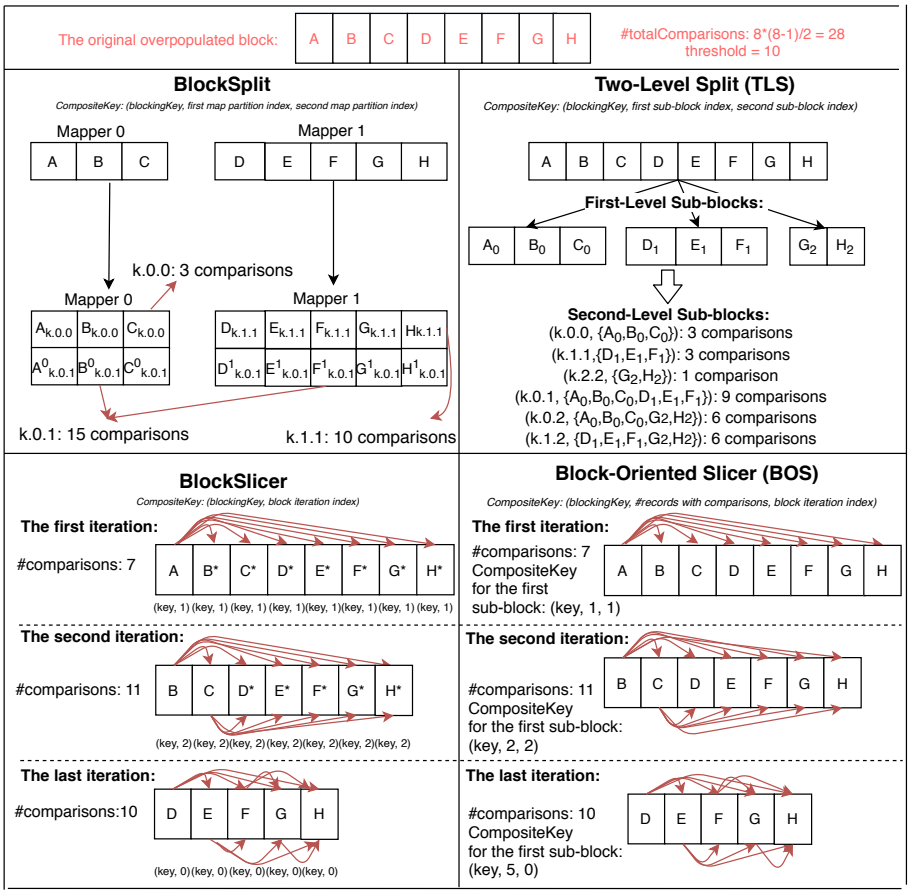


Figure 5.3: Example of the comparison rearrangement process

records in the overpopulated blocks into several sub-blocks. The second-level of sub-blocks is generated to keep a correct number of candidate pairs. For BlockSlicer and BOS, their splitting processes are iterated until the sub-block contains reasonable comparisons than the threshold. All four strategies are able to reduce the skewness. Theoretically speaking, BlockSplit, BlockSlicer, and BOS may still contain blocks whose number of comparisons exceeds the threshold value. With the help of proper reducer assignment strategies, a balanced reducer workload can be expected.

5.3 Evaluation

To evaluate the four strategies, we implemented them in Apache Spark. Next, we present our designed experiments and the corresponding results. In addition to the four introduced strategies, we consider the default load balancing strategy in Spark as well. In the first and second series of experiments, we evaluate how the runtime of different strategies is affected by the number of mappers and reducers. In the third series of experiments, we evaluate the robustness of different strategies when facing data with varied skew factors from 0 to 1. The last series of experiments evaluate and compare the speed-up of different strategies with increasing numbers of executors.

Table 5.1: Synthetic and real datasets.

Datasets	Name	#Records (mill.)	#Blocks	#Total Pairs (mill.)	#Max Block Pairs in %
Synthetic Datasets	$10^6 + 50\%$	1.5	21589	719	4.5%
	$2 * 10^6 + 05\%$	2.1	19263	1484	4.1%
Real Datasets	DBLP_0.5	0.5	8876	672	17%
	DBLP_1	1	11882	2570	16.4%

Table 5.2: Datasets used in the robustness experiments.

	Skew0	Skew0.2	Skew0.4	Skew0.6	Skew0.8	Skew1
#Blocks	1000	1000	1000	1000	1000	1000
#total Comps (mill.)	125	132	178	381	1180	3667
#Comps of Max Block Pairs (mill.)	0.12	1.27	11.52	87.95	521.66	2230.82
#Max Block Pairs in %	0.1%	0.9%	6.5%	23.1%	44.3%	60.8%

5.3.1 Experimental Setting

In this section, we introduce mainly the datasets that we used in different series of experiments. Our Spark YARN cluster used is the same one that was introduced in Section 4.3.1. For the pair-wise comparison step, Jaro-Winkler and absolute difference similarity functions are used to calculate similarities of strings and numbers separately. For the classification step, a threshold of 0.75 judges the match or non-match state of a pair, which is reasonable for both precision and recall. Each experiment was run three times, and the average result is reported.

Datasets for Experiments. We use both synthetic and real datasets for our evaluation, which are shown in Table 5.1. The synthetic datasets $10^6 + 50\%$ and $2 * 10^6 + 5\%$ are generated by GeCo, which has been introduced in Section 2.3.2. The naming scheme is the same as in Section 4.3.1. The real datasets are DBLP citation data, which is commonly used in related work. We first downloaded data in the XML format[2], and then choose two subsets of data forming two CSV files, with 0.5 million and 1 million records named DBLP_0.5 and DBLP_1, respectively. Records have four attributes, including title, authors, venue, and publication year. The blocking key for each dataset used in the first and second series of experiments is the first three letters of the attribute "title". For synthetic datasets, the blocking key is defined by concatenating the first two or three letters of Double Metaphone codes of attributes "given-name" and "surname". The corresponding number of blocks and comparisons, and the percentage that the largest block's comparison from the total comparisons can also be found in Table 5.1.

However, in all datasets shown in Section 4.3.1, the highest percentage of the largest block comparisons in the total comparisons is 17%, which is not a quite high level

[2]Download from https://dblp.uni-trier.de/xml/

Table 5.3: Imbalance Ratio of Different Strategies with 56 Reducers

	$10^6 + 50\%$	$2 * 10^6 + 05\%$	DBLP_0.5	DBLP_1
Default	25.049	25.108	29.137	26.747
BlockSplit	1.000	1.000	1.000	1.000
BlockSlicer	1.000	1.000	1.000	1.000
TLS	1.000	1.000	1.000	1.000
BOS	1.000	1.000	1.000	1.000

of skewness. To quantify the skewness and evaluate the robustness of each strategy under different levels of skewness, we modified the dataset DBLP_0.5 by appending directly a blocking key attribute, keep a constant number of blocks (1000 blocks for our experiments). The block sizes follow a Zipf distribution with skew parameter z (varied from 0 to 1 with a 0.2 interval), which means that the *kth* largest block has the number of records that is proportional to k^{-z}. The corresponding number of comparisons and the percentages of the largest block comparisons are shown in Table 5.2. It should be noticed that, the real skew factors for record pairs are even squared on the given skew parameter of records.

5.3.2 Evaluation of Different Numbers of Reducers

In this section, we evaluate how the four strategies along with the default load balancing strategy perform with the varied number of reducers. We first introduce the experimental design, then discuss the obtained results.

Experiments Design. For this series of experiments, we use all datasets shown in Table 5.1 to compare the five strategies. We recorded their total runtime, their overhead excluding the block distribution part[3] and their median GC time when performing the pair-wise comparison. We also calculated the imbalance ratio (IR) for the five strategies. Our full cluster resource (7 executors, each with 4 cores and 6G memory) is used, and we have 28 cores in total. What needs to be noted is that to achieve better parallelism, the number of mappers and reducers in all experiments are always the multiples of the number of total cores 28. For this series of experiments, the number of mappers is fixed with 28, and the number of reducers is varied from 56 to 224 (even multiples of 28) to see how they affect the runtime of each strategy as well.

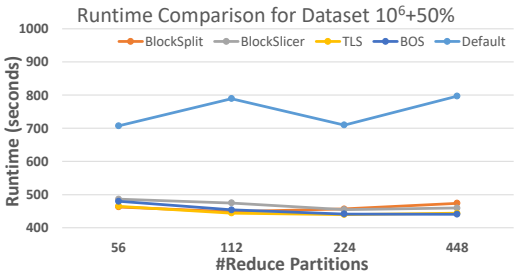
Results and Discussion: Table 5.3 and Table 5.4 show the IRs of five strategies when there are 56 and 448 reducers. For other reducer numbers, the IRs of different strategies are similar to 56 reducers (see Section A.3).

As we can see from it, with the default load balancing strategy, which does not concern about the imbalance between block sizes, their IRs are high, especially for the real datasets. This indicates a higher skew level for real datasets, which conforms to their percentages of the largest block comparisons from the total comparisons.

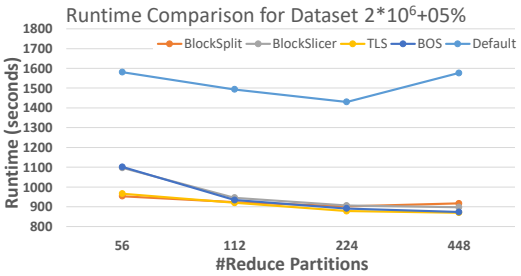
[3]Because the overhead for collecting the block distribution for different strategies is the same, we show here only the overhead excluding the pre-statistic part. It is calculated by subtracting the overhead of the pre-statistic part from the total overhead.

Table 5.4: Imbalance Ratio of Different Strategies with 448 Reducers

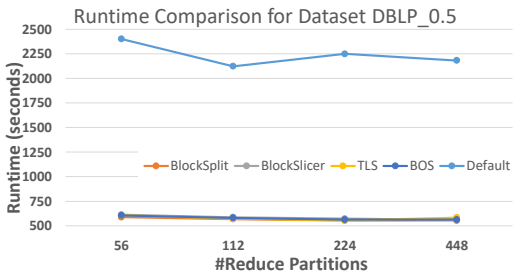
	$10^6 + 50\%$	$2 * 10^6 + 05\%$	DBLP_0.5	DBLP_1
Default	3.764	3.739	10.053	7.539
BlockSplit	1.000	1.000	1.000	1.000
BlockSlicer	1.000	1.000	1.000	1.000
TLS	1.000	1.000	1.004	1.003
BOS	1.000	1.000	1.000	1.000



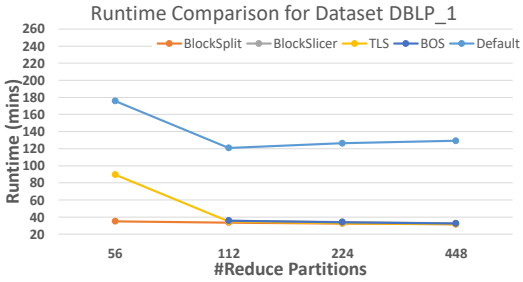
(a) Dataset $10^6 + 50\%$.



(b) Dataset $2 * 10^6 + 5\%$.



(c) Dataset DBLP_0.5.



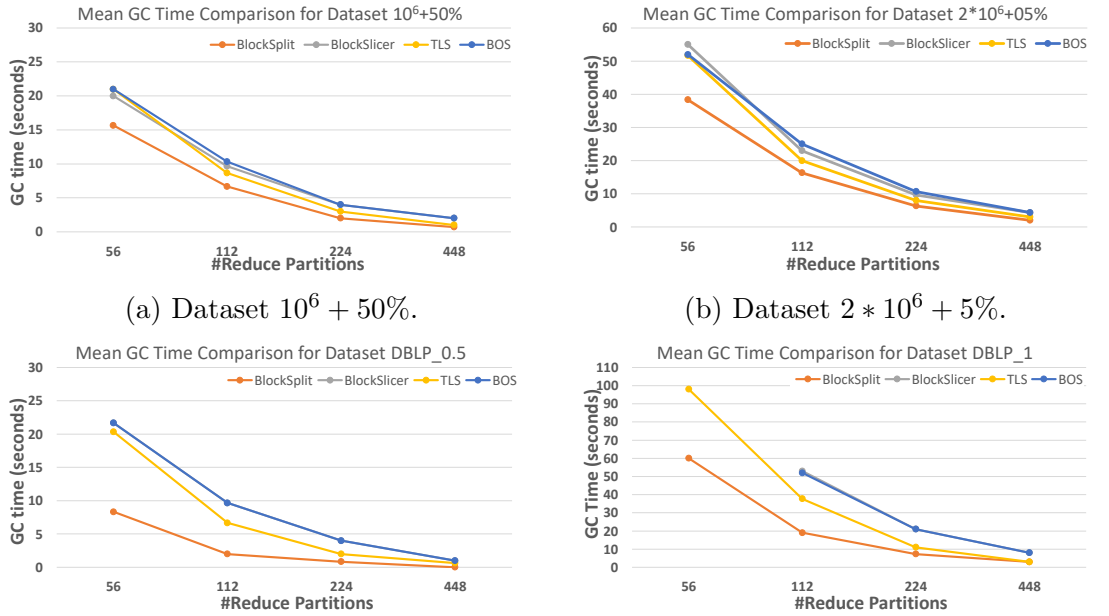
(d) Dataset DBLP_1.

Figure 5.4: Runtime comparison of different numbers of reducers.

With a higher number of reducers, the IRs can be reduced, because with more reducers, the extra workload of the reducer that contains the largest block is assigned to other reducers.

For the tailored strategies, with 56 partition reducers, all four strategies function quite good for all datasets and achieve almost optimal reducer workload balance, which should lead to same runtime for the computation time of the pair-wise comparison step. With 448 reducers, Blocksplit, BockSlicer and BOS function also almost optimal with IRs 1.000. TLS has higher IRs for datasets *DBLP*_0.5 and *DBLP*_1, but still lower than 0.5%, which did not obviously affect the runtime in our experiments.

Figure 5.4 shows the runtime comparison for the five strategies. Four tailored load balancing strategies reduce the runtime significantly compared to the default load balancing strategy. For datasets $10^6 + 50\%$ and DBLP_0.5, the runtime of four strategies is stable for all number of reducers. For the other two datasets with a larger number of comparisons, BlockSlicer and BOS have longer runtime with 56 reducers. Conjoining the results in Figure 5.5, for the recorded median GC time, which varies much for the different number of reducers on datasets $2 * 10^6 + 05\%$ and



(a) Dataset $10^6 + 50\%$.

(b) Dataset $2 * 10^6 + 5\%$.

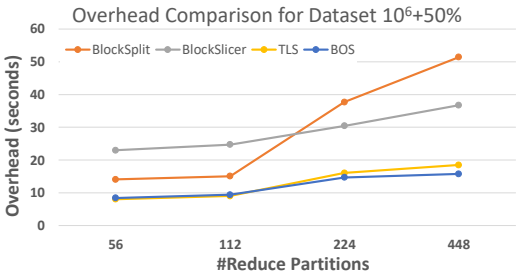
(c) Dataset DBLP_0.5.

(d) Dataset DBLP_1.

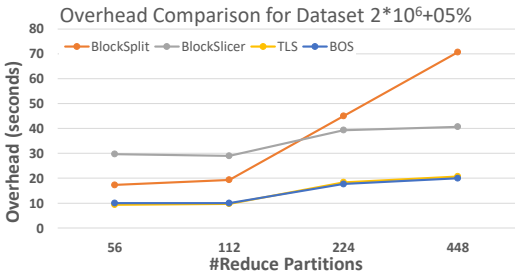
Figure 5.5: Median GC time comparison of different numbers of reducers.

DBLP_1, we can know the reason for the longer runtime for lower number of reducers is due to the required GC time. When the number of reducers is low, the threshold for identifying overpopulated blocks is higher, which still leaves many larger blocks after the splitting process and requires longer GC time for the comparisons of those large blocks. It should be noted that, the GC time only increases significantly when the memory is limited as our cluster. Among the four strategies, BlockSplit has the shortest GC time, because the overpopulated blocks are split first to 28 sub-blocks, leading to averagely smaller blocks, but more number of blocks. Our TLS has a longer GC time than BlockSplit, the reason is as follows. The overpopulated blocks are split into n sub-blocks based on their sizes, i.e., bigger blocks are split more. Because the percentages of the comparisons in the biggest blocks in the four datasets compared to the total number of comparisons are not quite high, n is smaller than the used number of mapper in BlockSplit. Thus, the average block sizes are larger than BlockSplit. However, BlockSlicer and BOS have a significantly longer GC time, even fail to run when the number of reducers is equal to 56 because they have more large blocks, whose number of comparisons is close to the threshold.

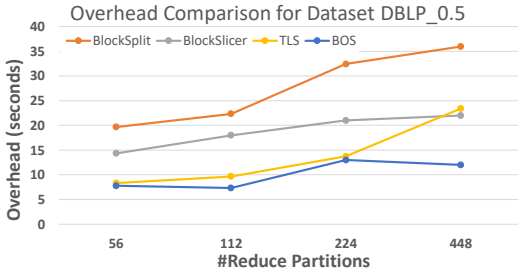
By excluding the GC time factor, from the results of the total runtime with a high number of reducers, our proposed two strategies TLS and BOS run fastest due to the low overhead excluding the block distribution part. Figure 5.6 shows clearly the overhead for each strategy. Our TLS and BOS reduce the overhead significantly. For the majority cases, BlockSplit has the biggest overhead, because it first splits any overpopulated block into a fixed number of sub-blocks (equal to the mapper), ignoring their sizes. Similar situation for TLS, compared to BOS, the final number of sub-blocks that an overpopulated block is split into is higher than BOS, which leads to higher overhead.



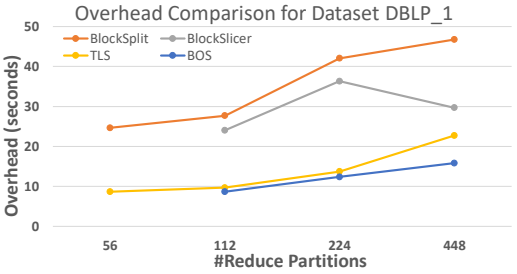
(a) Dataset $10^6 + 50\%$.



(b) Dataset $2 * 10^6 + 5\%$.

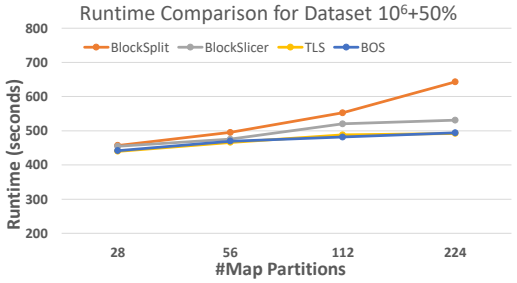


(c) Dataset DBLP_0.5.

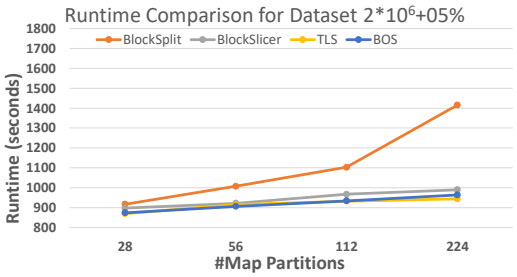


(d) Dataset DBLP_1.

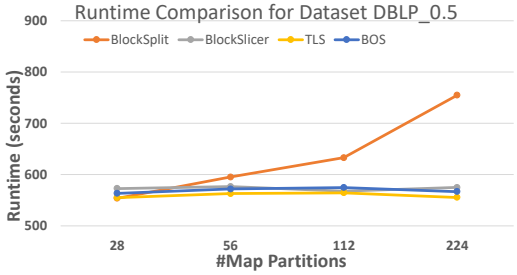
Figure 5.6: Overhead comparison of different numbers of reducers.



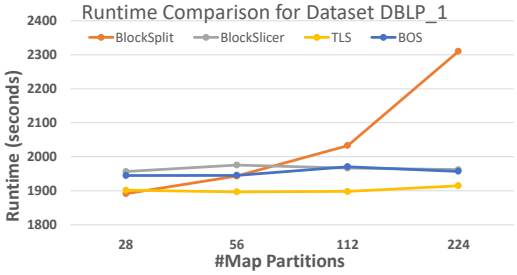
(a) Datasets $10^6 + 50\%$.



(b) Datasets $2 * 10^6 + 5\%$.



(c) Datasets DBLP_0.5.



(d) Datasets DBLP_1.

Figure 5.7: Runtime comparison of different numbers of mappers.

5.3.3 Evaluation of Different Numbers of Mappers

Experiments Design. For this series of experiments, we use still the four datsets shown in Table 5.1). We recorded all the total runtime of four strategies by using our full cluster resource (7 executors, each with 4 cores and 6G memory). Based on the results on the first serie of experiments, for datasets $10^6 + 50\%$ and DBLP_0.5, the number of reducers is 224. For the other two larger datasets, 448 reducers are

Table 5.5: Imbalance ratio of different strategies of robustness evaluation.

	Skew0	Skew0.2	Skew0.4	Skew0.6	Skew0.8	Skew1
Default	2.464	3.843	15.689	52.171	99.272	-
BlockSplit (Gathered)	1.120	2.159	14.480	51.686	47.566	22.525
BlockSplit (Shuffled)	1.120	1.005	1.0001	1.000	1.001	1.000
BlockSlicer	1.120	1.058	1.042	1.009	1.001	1.000
TLS	1.120	1.052	1.035	1.010	1.001	1.001
BOS	1.120	1.058	1.042	1.009	1.001	1.000

used. Then we varied the number of map partitions from 28 to 112 (even multiples of 28) to see how they affect the runtime of each strategy as well.

Results and Discussion: Figure 5.7 shows the runtime of different strategies with the increasing number of mappers. As we can see from it, the runtime of BlockSplit increases significantly, because each overpopulated block is replicated m times and m is equal to the number of mappers. This leads to a much higher number of sub-blocks from splitting the overpopulated blocks. The runtime of the other three strategies keeps stable because the records in one block are collected to a list, ignoring the mapper they come from. With 28 mappers, BlockSplit can provide competitive performance compared to the other three strategies, especially runs fast due to its lower GC time when the cluster resources are limited and the skew level is not quite high. With a higher number of map partitions, the other three strategies are more efficient.

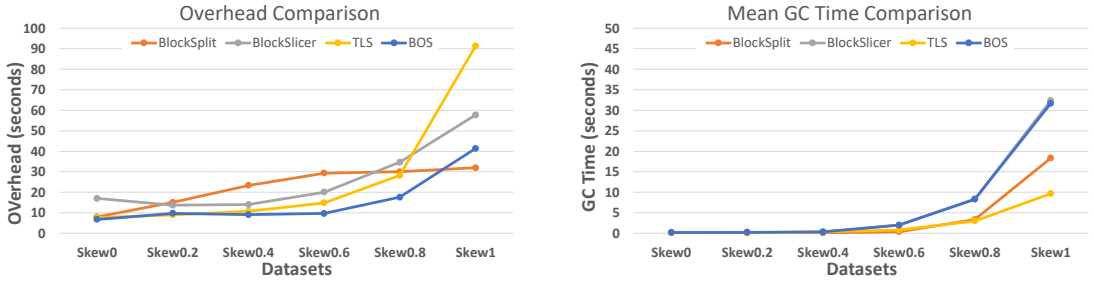
5.3.4 Robustness Evaluation

Experiments Design. For this series of experiments, we use the second group of datasets (see Table 5.2). It includes six datasets with different skew factors of the same base dataset *DBLP_0.5*. The skewness of the second group datasets is explicitly given to evaluate the robustness of strategies facing different skew levels. As we assign blocking keys following the Zipf distribution from beginning to end, records of one block are located together in the original skew datasets. Therefore, in addition to the original skew datasets, we shuffle all datasets randomly to make the records of one block dispersive. We recorded all the total runtime of five strategies on the six datasets by using our full cluster resource (7 executors, each with 4 cores and 6G memory) for both situations. For this series of experiments, we fixed 28 mappers and 224 reducers. As mentioned in, Chapter 2, for this series of experiments, since the number of record pairs varied notably much with different skew factors, we compare and report the average runtime of each strategy for completing one million record pairs for different datasets.

Results and Discussion: In Table 5.5, we show the imbalance ratio for five strategies in our robustness evaluation. As we can see from it, although blocks in dataset *Skew0* are in the same size, the imbalance ratio of the four tailored strategies for the reducer load is the highest 1.12. This happens because the greedy strategy

(a) Gathered skew datasets. (b) Shuffled skew datasets.

Figure 5.8: Runtime comparison per million records.

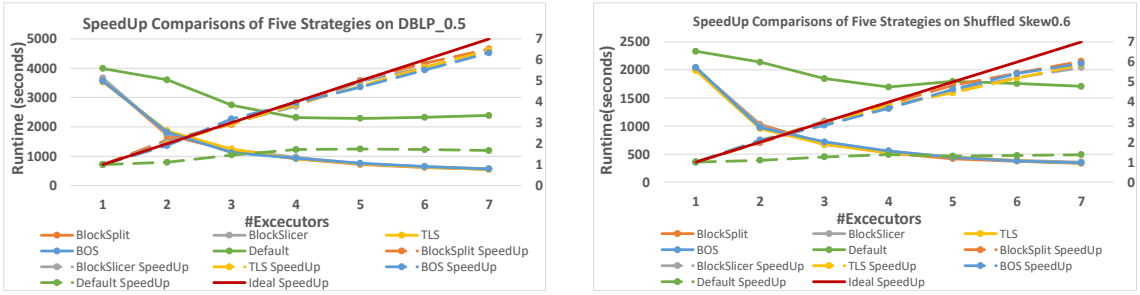


(a) Overhead comparison. (b) Median GC time comparison.

Figure 5.9: Overhead and GC time comparison of robustness evaluation.

to assign blocks to reducers. In total we have 224 reducer, after four rounds that 896 blocks are evenly assigned to reducers, the left 104 blocks are assigned to 104 reducers from the total 224 reducers, which leads to the imbalance. This reminds a possible factor that the correlation between the number and the size of the blocks, and the number of reducers for affecting the imbalance between reducers. Although for higher skew datasets, the imbalance ratio is lower than Dataset "Skew0", still much higher than the imbalance ratios we shown in Table 5.3 and Table 5.4 of the first series of experiments. This indicates that with fewer blocks, the complete balanced reducer load is more difficult to be achieved. Particularly, for the original skew datasets' group, in which the records of a block are next to each other and many of them are partitioned into the same map partition. This limits BlockSplit's ability to split large blocks. Other three strategies are not vulnerable to the input distribution and work still stably for this group. By removing this factor, BlockSplit is able to balance the reducer load with different level of skew, and even better than other strategies because the high number of sub-blocks it generates.

Figure 5.8 shows the average runtime of five strategies for completing one million record pairs on two groups of skew datasets. The difference is for BlockSplit, it cannot provide a balanced reducer load with the aforementioned reason. By excluding this factor, we discuss the evaluation results on the shuffled group in Figure 5.8b. For dataset *Skew0* and *Skew0.2*, the default strategy is still competitive to the other four strategies, because the skew is quite low and it has no overhead except running the real ER tasks. The default load balancing strategy can provide a relatively balanced reducer load under this situation. The skew-handling strategies start to improve efficiency from the dataset *Skew0.4*. Although the default strategy can be



(a) Dataset DBLP_0.5. (b) Dataset Skew0.6.

Figure 5.10: Speed-up comparison of different strategies.

run successfully except *Skew1.0*, due to the high skew, the average runtime is increased significantly. By contrast, all four strategies show their robustness facing different skew factors, the average runtime remains stable and is even reduced, because the negative effect that the overhead brings becomes insignificant with higher skew.

Figure 5.9a shows how the overhead time of different strategies excluding the block distribution part changes. As we can see from it, BlockSplit significantly increases its overhead time from dataset *Skew0* to dataset *Skew0,6*, because the number of sub-blocks significantly increases (from 1000 to 12340), and then remains relatively stable for the left datasets. The overhead excluding the block distribution part for BlockSlicer, TLS, and BOS becomes much longer for datasets *Skew0.8* and *Skew1*, because when a block is larger, and it is split to more pieces for these three strategies. With Zipf-distribution datasets, the time for splitting the largest block will dominate the splitting time, partitions with smaller blocks have to wait for the largest block to finish its split. This is unavoidable for these three strategies because we collect all records of a block together and the larger blocks are split into more sub-blocks for the sake of a balanced reducer load. This problem is the most typical for TLS because BlockSlicer and BOS generate the number of records, which is equal to the number of the first-level sub-block of TLS, while TLS generates much more sub-blocks in the second level of splitting. BlockSplit does not suffer from this problem, because all overpopulated blocks are considered non-distinctively. Records of each block are kept in each input map partition and are not collected together. As long as the records are well distributed in input map partitions, the skew problem does not exist for BlockSplit. According to the result, for a quite high skew factor, with the biggest block occupying more than 60% of comparisons, BlockSplit may have the highest efficiency when the cluster resource is not a bottleneck. However, if the cluster resources are limited as our case, as we can see from the result of dataset *Skew1* in Figure 5.8b, BlockSplit and TLS have close runtime. This is because the biggest block is not split into sufficient pieces, the longer GC time will devour the advantage the low overhead brings (see Figure 5.9b).

5.3.5 Speed-Up Evaluation

Experiments Design. For the last series of experiments, we use the dataset *DBLP_0.5* and Skew0.6. We varied the number of available executors from 1 to 7 to record the

corresponding runtime and calculate the speed-up. If the number of used executors is n, then we set $4n$ mappers and $32n$ reducers. As introduced in Section 4.3.1, in our Spark cluster, four executors out of seven have the slowest CPUs with a frequency of 2.00GHz, only one executor has a moderate CPU with a frequency of 2.27GHz, and the remaining two executors have the fastest CPUs with a frequency of 2.60GHz. When we only need a low number of executors, such as 1 or 2 executors, the runtime difference becomes nontrivial between using the slowest executors (with a frequency of 2.00GHz) and the faster executors (with a frequency of 2.27GHz or 2.60GHz). Since we cannot control which executor to use and our cluster has more than half slow executors, we drop the results for the situations with only 1, 2, or 3 executors to minimize the impacts caused by our heterogeneous cluster on the speed-up results. In this way, we collect the runtime and calculate the speed-up for different strategies.

Results and Discussion: Figure 5.10 shows the runtime and speed-up of five load balancing strategies on two datasets. The results of both datasets show a similar trend. As we can see from it, the default strategy can only be obviously accelerated from 1 to 3 executors, afterward, the runtime remains similar and is even increased from 3 to 4 executors. The speed-up from 1 to 3 executors is its ability to assign the extra workload of the reducer that contains the largest block to other reducers. However, afterward, the total runtime is dominated by the reducer with the largest block and cannot be reduced even when the executors are further increased. The increase from 3 to 4 executors should be caused by our heterogeneous cluster. For the other four tailored load balancing strategies, because all of them can balance the reducer load, as we can see from Figure 5.10, the speed-up is significantly improved compared to the default load balancing strategy, especially for the cases till 5 executors. For both datasets, a bigger difference between their speed-up and the ideal speed-up emerges for experiments with 6 and 7 executors, because of our heterogeneous cluster. The speed-up difference between the four tailored load balancing strategies is not obvious, which indicates a similar performance.

5.3.6 Evaluation Conclusion

As we can see from the above-introduced results, different factors can affect the performance of a block-splitting-based load balancing strategy, and it is often a trade-off between different factors. Therefore, there is no eternal winner under different circumstances. In the following, we conclude the our evaluation results.

First of all, because the splitting algorithm of BlockSplit heavily depends on the input map partitions, two obvious drawbacks have been observed. First, in the experiments of "different number of mappers" (see Figure 5.7), BlockSplit increases its runtime significantly, when the number of mappers increases, because each overpopulated block is splitted more times. With the best number of mappers, it can provide competitive performance compared to other three strategies. However, with higher number of mappers, other strategies are more efficient. Second, in the experiments of "robustness", BlockSplit loses its ability to balance the reducer load for the gather dataset, because a large amount of records from a same overpopulated block are assigned to a same mapper, which leads to the situation that still overlarged blocks are left.

Because the similarity calculation step occupies the majority of the total runtime, the most fundamental requirement for a block-splitting-based load balancing strategy is the ability to balance the reducer workload. To conclude results for IRs, all tailored strategies are sufficient to provide balanced workload (always better than 99.5% in Table 5.3, Table 5.4 and always higher than 95% in Table 5.5), the slight differences between their imbalance ratios cannot affect their runtime much in our experiments and is negligible. Among four strategies, except the bad case explained above for BlockSplit with gathered dataset, BlockSplit can provide the best or the same result for the vast majority cases. TLS has the highest possiblity to have a higher imbalanced load.

Another factor that might affect the runtime of the similarity calculation step is the required GC time, when the cluster resource is not sufficient for the current dataset. Based on the results in Figure 5.5 and Figure 5.9b, for the majority of datasets except Skew0.8 and Skew1, BlockSplit requires the shortest median GC time, because in those cases, BlockSplit splits overpopulated blocks into more pieces and also replicates each record of a overpopulated block more times than other strategies, leading to smaller sizes of rearranged blocks. For datasets Skew0.8 and Skew1 with 224 reducers, TLS requires less GC time than others, because the overpopulated blocks are split the most time with TLS, the newly-obtained blocks after splitting are the smallest, leading to the lowest GC time.

The remaining factor that can affect the total runtime of ER is the overhead required due to the tailored load balancing strategies. Because in our implementation, for all four strategies, we commonly use the BDM, in the evaluation, we consider the required excluding the block distribution part. It should be noticed, for collecting block distribution, actually, only BlockSplit require the information of contained records in mappers, the other three strategies do not. From this point of view, BlockSlicer, BOS and TLS should be more scalable than BlockSplit. For the required overhead excluding the block distribution part, BOS and TLS require less for datasets except dataset Skew1 than BlockSplit and BlockSlicer (see Figure 5.6 and Figure 5.9a). The reason behind is our proposed block-wise key assignment instead of record-wise assignment of BlockSplit and BlockSlicer. For the special case with dataset Skew1, TLS requires particularly long overhead due to the assumed zipf distribution of the dataset. The splitting of the largest block consumes quite high time for TLS, and this process cannot be parallelized. For a common dataset, there should be multiple overpopulated blocks in similar sizes instead of only one extremly large block in dataset Skew1, TLS's overhead should still have a lower overhead.

5.4 Related Work

In this section, we discuss the related work regarding load balancing strategies. We start with several general load balancing strategies for various applications, then narrow the discussion to load balancing strategies for joins and ER problems.

Load balancing strategies with general purposes in MapReduce: The load imbalance problem has been exposed in a good deal of research. It can be caused by different types of reasons, which are overviewed and classified by Kwon et al. [2011].

Therein, partitioning skew is the most common case, which means the reducers are allocated biased loads due to the different complexities of map outputs. The following research handles this problem in different directions. Several studies first collect the key frequencies and then propose their own strategies to assign map output tasks to reducers as balanced as possible [Gavagsaz et al., 2018; Ibrahim et al., 2010, 2013; Sherif and Ngomo, 2015; Tang et al., 2016]. Ibrahim et al. designed their strategies not only depending on the key sizes as the most research considered but also the localities of map output tasks [Ibrahim et al., 2010, 2013]. Focusing only on getting a global key distribution, Gufler et al. [2012] proposed algorithms to get a global histogram in the distributed environment. To make the data profile more scalable, Yan et al. [2013b] proposed to use an approximate data structure, i.e., sketches, to gather the statistics of keys.

In addition to the above-mentioned research, which only develops strategies for distributing map output tasks to reducers based on the key frequencies, several researchers [Chen et al., 2014; Gufler et al., 2011; Qi et al., 2015; Ramakrishnan et al., 2012; Tang et al., 2018; Xu et al., 2012] also proposed to split the overpopulated map output tasks to subtasks so that even facing quite a heavy skew, the loads for reducers can still be relatively balanced. However, all strategies are quite general by simply chopping a map output task into several pieces and assigning them to multiple reducers. However, for some complex operations, in which the records need to operate with all other records from one original map output task, the simply chopping does not work and it cannot provide a consistent result as before. Because the operations between the records, which are originally placed in the same map output task and after chopping are placed in different map output subtasks, are missing (operations are not performed cross different subtasks).

Load balancing strategies for joins: To solve the partitioning skew for reducers of joins, Atta et al. [2011] implemented two types of range partition algorithms originally proposed in [DeWitt et al., 1992]. Based on sampling results, if there are too many tuples for one join attribute value, those tuples are redistributed to multiple reducers. To keep consistent join results, replication is required. However, in this paper, the strategies only simply replicate those tuples from the first relation n (the number of reducers the tuples should be redistributed to) times and make each reducer hold all tuples from the first relation and a subset of tuples from the second relation, which generate too many replicas.

Okcan and Riedewald [2011] propose an algorithm called 1-bucket-theta for theta joins by using a join matrix to explore the best matrix-to-reducer mapping, which aims to minimize the replicates of tuples and meanwhile balance input and output costs of reducers. However, this algorithm works near-optimal for the Cartesian product but not for theta joins. To make up the deficiency m-bucket is proposed to improve the efficiency of 1-bucket-theta for theta joins. Hassan et al. [2014] proposed the MRFA-join algorithm, which is based on distributed histograms and a randomized key redistribution approach. However, they also replicated all tuples from one of the two relations.

For another perspective of solutions, Li et al. [2017] and Kwon et al. [2012] do not predict the key frequencies in advance but proposed to monitor the states of

reducers and dynamically transfer a part of loads from busy reducers to idle ones if both states of reducers exist at the same time.

Load balancing strategies for ER: Similar to join operations, ER is also a task, for which simply splitting overpopulated blocks is not possible because any record needs to compare to all other records in the same block. Particularly, ER involves more complex calculations for reducers than a normal join operation, the complexity or the imbalance degree is not determined by the size of the key, but by the number of comparisons. Therefore, it needs an elaborate design of the splitting algorithm.

To provide a more balanced solution for block-skewed MapReduce-based ER, there have been many strategies proposed. In addition to the strategy BlockSplit, Kolb et al. [2012b] proposed another strategy called PairRange. It enumerates all comparisons, then all comparisons are divided into ranges of the same size and assigned to reducers. This can provide more balanced loads but the overhead is larger. BlockSlicer Gomes Mestre and Pires [2013] has already been discussed and compared in our research.

To make the load balancing algorithms more scalable, Yan et al. [2013a] replace BDM with a sketch-based approximate data profiling method for the sake of better scalability. However, it is at the cost of the balance of reducers.

Hsueh et al. [2014] focused on blocking-based ER with multiple keys, the load imbalance is solved in a way similar to PairRange. McNeill et al. [2012] solve the load imbalance caused by skewed blocks from another perspective. When oversized blocks are detected, more concrete blocking keys are generated on the oversized blocking key, which prevents from oversized blocks. Karapiperis and Verykios [2015] permute all comparisons and then assign them to reducers in a round-robin way.

5.5 Summary

In this chapter, we discuss and compare four block-splitting-based load balancing strategies for parallel ER. They eliminate the overpopulated bottleneck blocks by splitting them into sub-blocks. For two state-of-the-art strategies BlockSplit and BlockSlicer, we analyze them in-depth and point out their advantages and disadvantages theoretically. Based on the analysis, we propose two other strategies TLS and BOS. We implemented and evaluated them with our Spark cluster. The results show that all four strategies can solve the reducer imbalance problem, at the cost of extra overhead for collecting block distributions and splitting the overpopulated blocks. Under limited cluster resource, their runtime may be increased due to the required long GC time, if a low number of reducer is set. However, our two strategies TLS and BOS reduce the required overhead by directly assigning the composite key to each block instead of each record and improves efficiency for parallel ER in the majority of experiments. For a block-splitting-based load balancing strategy, the fundamental goal is a balance reducer load. Because compared to the time of performing the similarity calculation step, the overhead required for collecting block distributions and splitting the blocks is relatively small. In the premise of balanced loads, a splitting strategy that supporting assigning the composite key to each block can reduce the overhead and improve efficiency. For a cluster with limited memory resource, for datasets, whose records of the same block are not close to each

other, BlockSplit is a good option, as the blocks after splitting are averagely smaller than those in other strategies. In the second place, TLS requires less memory than BlockSlicer and BOS, meanwhile it does not rely on a good distribution of input records.

6. Hybrid Similarity Calculation Approach for Entity Resolution

> This chapter shares material with the BTW'19 paper "The Best of Both Worlds: Combining Hand-Tuned and Word-Embedding-Based Similarity Measures for Entity Resolution" [Chen et al., 2019a].

As introduced in Chapter 2, under the challenges big data brings, four research goals have been identified for Entity Resolution (ER), they are efficiency, scalability, effectiveness, and less human effort involved. In Chapter 4 and Chapter 5, we presented our research for parallel ER when ER faces the data volume challenge, whose main goals are to improve its efficiency and scalability. However, big data does not only mean data is "big", but it also signifies variety. In this chapter, we switch our focus to the other big data character that we concern in this thesis: data variety, which leads our research to approach the other two goals - effectiveness and less human effort involved.

Data variety means that data comes from diverse domains, and is with various forms and properties. Facing data variety, we observe that two research challenges are raised for ER. One challenge is that it is more difficult to select suitable traditional similarity measures when we are unfamiliar with the input data. When ill-suited similarity measures are adopted, the effectiveness of ER will be reduced. Recently, word embedding has been proposed to assist in calculating similarities for ER. Different from the non-commonality of traditional similarity measures for various data, it uniformly maps words of different types of data into vectors using a neural word embedding model and is also able to grasp the semantics of words to identify possible duplicates that do not look similar. Research [Ebraheem et al., 2018; Kooli et al., 2018; Mudgal et al., 2018] has shown the effectiveness benefits, when word embedding based approaches are used to calculate similarities between records, especially for cases that values of input data contain semantics. However, adding an extra word embedding step to the similarity calculation phase will probably have negative effects on efficiency. For instance for attributes without semantic meanings, especially attributes with numerical values, it makes little sense to map them

to a vector space using word embedding because distances between them cannot be correctly managed due to the missing semantics. Still, numeric data play an important role in expressing records in financial business data. Therefore, in this section, we propose to combine traditional hand-tuned and popular word-embedding-based similarity calculations for ER, always choosing the most suitable similarity measures for each attribute to achieve the best accuracy for a given input dataset. Specifically speaking, concerning attributes with high appropriateness of word embedding techniques, word embedding is firstly applied to map the data to vectors using FastText pre-trained model, then cosine similarity is used to calculate similarities between record pairs. For other attributes, for which word embedding is not suitable, particularly numerical values, a tailored similarity function is used. We summarize our contributions as follows:

- We propose a hybrid similarity calculation method for ER: the attribute-based selection between traditional hand-tuned similarity functions and word-embedding cosine similarity function to achieve higher effectiveness;
- We identify attributes that suit word embedding more than traditional similarity measures, which guides choosing the most suitable similarity measure for ER;
- We design different combinations of adopted similarity functions and evaluate them on both real and synthetic datasets[1]. Our results show that the hybrid solution can outperform other solutions that purely use traditional or word-embedding-based similarity calculations when the attributes of a dataset contain both semantic and non-semantic attributes. Based on the results, we conclude the way for the combination and the confusing attributes that need careful considerations.

The rest of this chapter is organized as follows: We firstly introduce our hybrid method in Section 6.1 and show the evaluation result in Section 6.2. Then we present related work in Section 6.3. At last, we conclude our work in Section 6.4.

6.1 Our Hybrid Approach for Entity Resolution Similarity Calculation

In this section, we introduce our ER process using hybrid similarity calculation methods. To be noticed, the goal of our research is to explore the impact of using different combinations of traditional or word embedding based similarity measures on effectiveness. Hence, we do not take blocking (indexing) techniques into consideration to our process to improve efficiency. With this, we get the most exact results since by blocking, we might block away matches in the worst case.

Figure 6.1 shows the entire process. The process first asks the user whether there is an available property file, which means the user knows the data quite well and knows

[1]The source code and the datasets can be found at https://git.iti.cs.ovgu.de/Chen/entity-resolution-for-big-data.

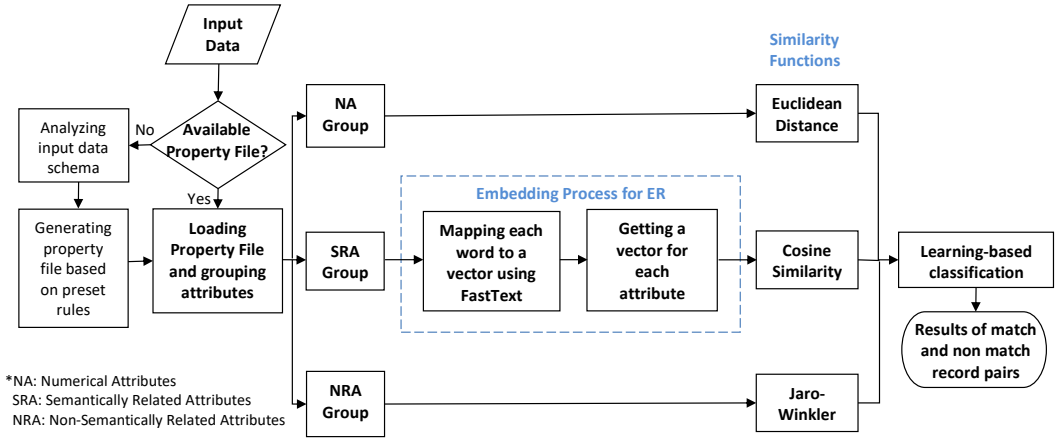


Figure 6.1: Flowchart of our hybrid method.

the best choices of similarity functions to compare each attribute (excluding identifiers). If this is the case, we directly load the property file and divide all attributes into different groups based on the similarity measures defined in the property file. Otherwise, a property file will be generated based on the input data schema and our preset rules. To achieve satisfying accuracy and provide guidance to choose suitable similarity measures, we classify common attributes into three groups, propose suitable similarity measures for each group and then the corresponding property file can be generated:

Numerical Attributes (NA): Numerical attributes refer to those attributes, whose values can be compared, for example, the age. In this work, we use Euclidean distance as an example to provide reasonable accuracy for numerical attributes. However, not all numbers belong to this group. In contrast to numerical attributes, the other type of numbers are named as numerical strings, which act like strings and their values cannot be compared to be bigger or smaller. For example, telephone numbers or postcodes are considered as numerical strings. All numerical strings belong to the next group: non-semantically related attributes.

Non-semantically Related Attributes (NRA): These kinds of attributes are those whose values are without semantic meaning (e.g., people names or numerical strings) and their lengths are usually short. In ER tasks, the different values of this type of attributes of a matching pair are usually caused by typos and formats. Using word embedding for this case may consider those values with a big distance, which can lead to lower accuracy. Therefore, for this group, we propose to use Jaro-Winkler similarity functions instead of word embedding. Moreover, Jaro-Winkler has a higher speed than the other traditional functions, such as Levenshtein similarity.

Semantically Related Attributes (SRA): The last group is semantically related attributes, whose values are strings with various meanings (often long), multiple strings, or even whole sentences. For this group, the word-embedding plus cosine similarity approach is chosen to calculate the similarity. Traditional syntactical-based similarity measures are not used for semantically related attributes, because they only check the edit distances between values of attributes, and are not able

to realize the meaning or the distributional semantic behind so that they would probably provide unacceptable results.

In terms of the predefined rules, we can calculate the similarity of each corresponding attribute pairs (cf. Section 6.1.1). Afterward, the classification step is performed to divide all pairs into matching and non-matching groups (cf. Section 6.1.2).

6.1.1 Attribute Similarity

In this subsection, we specify how the similarity of each pair of attributes is calculated by the preset rules.

NA and NRA Similarity. According to the two rules regarding NA and NRA, the similarity between two attribute values of records calculated straightforwardly with Jaro-Winkler similarity and Euclidean metric, because the granularity for them is the entire attribute, which is calculated with the following formula:

$$attrSim(r_1.attr, r_2.attr) = \begin{cases} Euclidean(r_1.attr, r_2.attr), & attr \in NA; \\ Jaro\_Winkler(r_1.attr, r_2.attr), & attr \in NRA. \end{cases} \tag{6.1}$$

SRA Similarity. Inspired by the work of [Bojanowski et al., 2016], we use FastText, an extension of the continuous skip-gram model [Mikolov et al., 2013b] that is used to produce word embeddings (i.e., vectors), to obtain the similarity of semantically related attributes. The FastText model is chosen instead of word2vec and GloVe because use cases will range across diverse domains. Hence, we cannot guarantee coverage with only models on word-level. Given an SRA, the following three steps are performed to get the final similarity value between two attribute values:

- *Vector representation of words.* The first step is to get the vector representation of each word in the attribute. The input of the used model FastText is a text corpus. Given enough text data and contexts, FastText can achieve highly accurate meanings of the words appearing in the corpus and establishes a word's association with other words. The output is a set of vectors, that is, words in the corpus are transformed to a vector representation in a semantic vector space. Moreover, FastText is capable of predicting the vector representation for words not occurring in the corpus, since it decomposes each word into a set of characters of n-grams. Hence, every word appearing in the attribute value can be converted into a vector representation $\vec{w}$.

- *Vector representation of attributes.* Afterwards, considering that there may be two or more words in one attribute, we achieve the vector representations of attributes by computing the mean value of all the word vectors in the attribute:

$$\vec{attr} = \frac{\sum_{i=1}^{n} \vec{w_i}}{n}, \tag{6.2}$$

where n is the number of words in attributes.

Table 6.1: Datasets used in experiments.

Datasets		#Pairs (#DS1&#DS2)	#Matches	#SRA	#NRA	#NA
Real Datasets	DBLP-ACM	6001104 (2616&2294)	2224	2	1	1
	Amazon-Google	4400264 (1364&3226)	1300	3	0	1
Generated Dataset	Persons	551250 (1050&1050)	96	2	6	5

- *Cosine similarity to compute the similarity value.* After we get the two vector representations of two attribute values, the last step is to use cosine similarity measure to get the final similarity value between two attribute vectors. More formally, the similarity of SRA is represented by:

$$attrSim(r_1.attr, r_2.attr) = cosine(r_1.\vec{attr}, r_2.\vec{attr}), \quad attr \in SRA. \tag{6.3}$$

6.1.2 Learning-Based Classification

After the similarity for each attribute pair is calculated with their respective methods, a classification step is used to classify pairs to matches or non-matches. As mentioned above, the simplest approach is a threshold-based method, which compares the average similarity score with the preset threshold value. Those pairs whose average score is higher than the threshold are considered as matches, and vice versa. However, to get the average score, similarity scores for each attribute are firstly summed up, which loses detailed information contained in the separated attribute similarities. Therefore, in our ER process, we adopt a learning-based classification step to overcome this drawback. A learning-based classification method firstly trains a classifier on a training dataset with available matching or non-matching labels, then the trained classifier is used to classify pairs with match or non-match status. So far, there have been different classifiers proposed. For the experiments described in Section 6.2, we employ the following three classifiers to compare the ER effectiveness results by using different combinations of traditional or word embedding based similarity measures: Tree Boosting (XGBoost), Random Forest (RF), and K-Nearest Neighbor (KNN). These three classifiers are employed as examples, which are intended to represent general solutions for classification in an ER context.

6.2 Evaluation

In this section, we show our designed similarity calculation methods and the evaluation of their F-measure on different real-world datasets and synthetic datasets. Next, we show the datasets used for our experiments and represent designed combinations of similarity calculation methods.

6.2.1 Datasets Used

Table 6.1 shows the three datasets we used for our experiments.

The first dataset "*DBLP-ACM Citation*" includes two parts, which are from the DBLP citation database and the ACM citation database, respectively. All of them have four attributes, including title, authors, venue, and publication year. Therein, "title" and "venue" are considered as SRAs, while "authors" is NRA and "publication year" is NA.

The second dataset "*Amazon-Google Products*" includes two parts as well, which are from Amazon and Google. Both of them have four attributes, including id, name, description, manufacturer, and price. Based on their properties, "name","description", and "manufacturer" are considered as SRAs, while "price" is NA. All above datasets are downloaded from [Rahm, 2017], which are benchmark datasets often used for ER.

The last dataset is *synthetic* and generated by the data generator GeCo introduced in Section 2.3.2. As introduced there, the generated dataset contains personal information with the following 13 attributes: gender, given-name, surname, postcode, city, sex, telephone-number, credit-card-number, income-normal, age-uniform, income, age, and blood-pressure. Therein, the last five attributes are considered as NAs, while "sex" and "gender" are SRAs and the other attributes are NRAs.

The first two real datasets are commonly-used benchmark datasets for ER. However, both of them only contain one numerical value, which we think the most useful attribute type to show differences in using different combinations of similarity calculations. Therefore, we involve the generated dataset as well, which contains five attributes with numerical values. Due to privacy issues, we are not able to have real personal or business data, which contains several attributes with numerical values.

6.2.2 Design of Different Combinations of Similarity Calculation Methods

To evaluate our proposed hybrid similarity calculation approach and compare the effectiveness between approaches using different similarity measures, we design the following three combinations of similarity measures for different types of attributes:

- *Traditional hand-crafted similarity functions only.* For this scenario, we use the Jaro Winkler similarity function to calculate the similarity of all string attributes, i.e., NRAs plus SRAs, and use Euclidean distance to calculate the similarity of all NAs.
- *Word embedding and cosine similarity based method only.* For this scenario, we use word-embedding-based method for all attributes.
- *Our proposed hybrid method.* We propose to use word embedding for SRAs, Jaro Winkler for NRAs, and the Euclidean distance function for NAs.

We implemented the aforementioned three classifiers with the scikit-learn library. Results are reported for a random split of 66/34% training/test data, each including respectively 66/34% of the existing match and non-matches.

Table 6.2: Evaluation results with different classifiers (F-measure).

Combinations		XGBoost	RF	KNN
Generated Dataset	Traditional	100	100	88.46
	WordEmbedding	100	100	100
	Hybrid	100	100	46.15
DBLP-ACM	Traditional	97.04	97.70	95.17
	WordEmbedding	92.56	94.82	93.94
	Hybrid	93.69	94.28	89.31
Amazon-Google	Traditional	20.19	25.35	21.11
	WordEmbedding	19.10	31.09	24.10
	Hybrid	29.72	38.32	19.78

6.2.3 Results and Discussion

Table 6.2 shows the results of our evaluation. Next, we will discuss the result of each dataset in detail.

Generated dataset: As can be seen, the F-measure reaches its optimal for the generated dataset, with word embedding solutions performing consistently well across classifiers. This is a slightly surprising finding, as most attributes can be labeled as non-semantic. We deem the goodness of embeddings to be partly dependent on the coverage of the FastText learned representations. The purpose of using the generated dataset is it contains several NAs, which is promising to show the accuracy difference between different approaches. However, the generated dataset still lacks unpredictability and complexity compared to a real dataset, so that all F-measures are very high and we cannot get valuable conclusions from it. We also observe that the results of KNN are affected by the existence of irrelevant features, whereby the similarity of items on one dimension leads to misclassifications for close neighbors. For its less effective configurations, the model outputs 6 and 21 false positives out of 30, reducing the F-measure on this dataset.

DBLP-ACM dataset: For the relatively clean and easy citation dataset, the F-measure is observed to be consistently higher than 95% with traditional approaches outperforming others. The word-embedding-based and the hybrid approach show lower results. By careful consideration of its attributes properties, we found that the "title" attribute is, although having long strings, actually not semantic related. A paper title normally only has one version and its name usually only differs due to possible typos. Under this situation, word-embedding-based methods may fail. Therefore, for those attributes with long strings, careful consideration is needed about whether an edit-distance-based or semantic-based similarity measure is more suitable for them. The inadequate use of embeddings for the year attribute might have hindered the quality of the learning process.

Amazon-Google dataset: For the more complex product dataset (various descriptions may express the same semantic meaning) we observe that hybrid solutions outperform the other two pure solutions, and achieve the best results with XGBoost and RF. For KNN we also observe some problems with the existence of irrelevant dimensions, which increase the number of false negatives. Overall, embedding-based

Table 6.3: Example of a matching pair.

(a) Original data.

Id	Name	Description	Manufacturer	Price
Amazon b00005rd70	train sim modeler design studio	with train sim modeler you can create 3d...	abacus	39.99
Google 1769133250 3246508957	abacus train sim modeler	microsoft train simulator brings the most...		29.84

(b) Similarity scores.

Id	Name	Description	Manufacturer	Price
Traditional	0.661	0.721	0.0	0.999
Word Embedding	0.857	0.872	0.0	-0.036
Hybrid	0.857	0.872	0.0	0.999

approaches outperform traditional solutions (for classifiers RF and KNN) or provide a comparable result with XGBoost, which indicates the necessity to use word embedding for datasets with real semantic attributes. The hybrid approach performs the best with XGBoost and RF, which indicates that by carefully choosing word-embedding-based or traditional similarity measure according to attribute properties (mainly basing on non-semantic and semantic) a better accuracy can be achieved than using only one of them. Table 6.3 shows an example of the similarity scores of a matched pair. For the numerical attribute "price", the similarity score between "39.99" and "29.84" is even a negative value, which indicates an opposite meaning of two values and is not true. Therefore, we can observe that embeddings create a strong signal for attributes with common semantics, but provide little help, even negatively affect the similarity calculation of attributes lacking such properties, especially for numberical attributes.

We should also note that the F-measure values we report for this challenging dataset are much lower than many published results. We attribute this to the fact that almost all previous research adopts either blocking or thresholding techniques to reduce the amount of non-matching pairs, so that the training data set is much more balanced, helping to achieve a higher F-measure. For this paper, our purpose is solely to test the benefit of our proposed hybrid similarity calculation approach. Therefore, we avoided blocking and thresholding, to highlight the specific impact of the approaches.

To conclude our discussion, a word-embedding-based approach works predominately better for semantic attributes. For non-semantic attributes, it may also be possible to achieve an F-measure which is comparable (i.e., as seen for the generated dataset) or worse (i.e., as seen for the DBLP-ACM dataset) than traditional similarity measures. However, for numerical values, word embedding is not recommended since a hybrid approach shows obviously better F-measure than only using word embedding

(based on the result of Amazon-Google dataset with XGBoost and RF classifiers). Therefore, the safest way for similarity calculations of ER problems is to use word embedding for semantic attributes and to use traditional similarity measures for non-semantic attributes. Besides, interestingly, in our findings, we note that the choice of classifier seems to be secondary to the choice of similarity measures, with simple classifiers being able to outperform more complex ones, provided they are given adequate (i.e., descriptive, distinctive) similarity measures as input. We expect this to be partly explainable from the simplicity of the datasets (for the first two cases) and the improvements brought by semantics (in the third case).

6.3 Related Work

In this section, we discuss the related work on calculating similarities for record attributes in an ER process. At first, we introduce several surveys, which conclude mainly traditional similarity measures for strings. Then we present the approaches using word embedding to solve ER tasks. At last, we show the research approaches, which, similarly as ours, propose hybrid similarity calculation approaches for ER.

Traditional similarity calculation for ER: As introduced in Chapter 2, there have been numerous similarity measures for strings developed so far. Related surveys can be found in [Brizan and Tansel, 2006; Koudas et al., 2006; Newcombe and Kennedy, 1962]. The involved similarity measures are mainly traditional, syntactic-based approaches. Cohen et al. [2003] compared different string similarity metrics for name-matching tasks, including Jaccard, Jaro, Jaro-Winkler, Levenstein, SmithWaterman, Jensen-Shannon, TFIDF, Monge-Elkan, and softTFIDF. Monge-Elkan and softTFIDF are grouped into hybrid distance functions, since they provide strategies to combine similarities of subsets to get a final result and the similarities of subsets are calculated with secondary distance functions.

Word embedding for ER: As aforementioned, recent research has considered applying word embedding for ER. There are two main research questions of employing word embedding for ER. One is which embedding granularity to use for ER; the other one is how to get the vector of each attribute of a tuple after each word or sub-word has been embedded.

Ebraheem et al. adopt word-level embedding using the GloVe pre-trained dictionary [Pennington and R. Socher, 2014] and propose two methods to get the vector representation of an attribute: an averaging method and an RNN-based method with LSTMs. Then a representation of a tuple is obtained by concatenating the vectors of all its attributes [Ebraheem et al., 2018].

Kooli et al. employ N-gram-level embedding using the Fasttext library and then concatenate all vectors of all subwords [Kooli et al., 2018]. N-gram-level embedding should provide a more accurate result when there is a large proportion of infrequent words in the input dataset [Mudgal et al., 2018]. For ER tasks, data is often dirty and contains many infrequent words, therefore, in our work, we also use N-gram-level embedding.

Mudgal et al. study several possible embedding choices from both the granularity of the embedding and adopted model, and sketch a design space for deep learning solutions [Mudgal et al., 2018].

Hybrid similarity calculation for ER: There have been research approaches, which use more than one similarity measure to calculate the similarities between records. Köpcke et al. [2010] combined different similarity functions for non-learning and learning based ER. They found that for non-learning based ER, using a single similarity function on only one attribute provides the best effectiveness in most cases. While for learning-based ER, using several similarity functions for each attribute and combining similarity results from two attributes can provide better results, because the classifiers are able to find the underlying effective features by providing different options with multiple similarity functions. The above-mentioned comparison study [Cohen et al., 2003] also combines different similarity functions to provide abundant features for classifiers. Their result also shows that combined measures outperform the individual metrics.

The existing hybrid similarity calculation approaches are mainly using multiple similarity functions for one attribute and considering similarities of different attributes to provide broader perspectives for classifiers and further improve effectiveness. Our hybrid similarity calculation method is different from the above hybrid approaches. Our focus is to recognize the limitation of using word embedding based approaches for numerical and non-semantic attributes and choose suitable similarity measures based on the characteristics of different attributes.

6.4 Summary

In this chapter, we propose to use a hybrid similarity calculation solution for ER tasks and provide a practical evaluation of three different combinations of similarity measures with general machine learning classifiers. We find that embeddings are generally useful, though they are not a silver bullet, and both hybrid and traditional approaches can achieve superior results. We find that similarity measures can have a greater impact than the choice of classifiers in the resulting goodness of an ER process, with simple classifiers being able to outperform more complex ones, provided they are given adequate (i.e., descriptive, distinctive) similarity measures as input. Our observations concur with other results in the literature (e.g. [Mudgal et al., 2018]) in the conclusion that to build general ER systems it is necessary to provide room for adaptive configurations impacting all steps of the ER process. In summary, in our research, we apply a prototypical workflow for ER, which does not involve blocking or thresholding and performs a learning-based classification step with general classifiers. Based on the evaluation results, we show that the current use of word embeddings alongside traditional measures for ER opens-up a bundle of promising choices for practitioners, without lending itself easily to a one-size-fits-all solution.

7. Heterogeneous Committee-Based Active Learning for Entity Resolution

This chapter shares material with the ADBIS'19 paper "Heterogeneous Committee-Based Active Learning for Entity Resolution (HeALER)" [Chen et al., 2019b].

In Chapter 6, we focused on the pair-wise comparison step of Entity Resolution (ER) and discussed the effectiveness of using different combinations of similarity calculation approaches, which is the first part of research under the data variety challenge. Therein, we mentioned that although a threshold-based classification method is efficient and easy to apply, detailed information contained in the separated attribute similarities may be lost in the process of summing up similarity scores to calculate the average score. Therefore, in many cases, learning-based classification is required to improve the classification result and the effectiveness of ER.

Facing various data, the absence of training data becomes another challenge. A large amount of data needs to be labeled by domain experts to ensure satisfactory effectiveness. This kind of human effort is quite expensive and may limit the adoption of Machine Learning (ML) approaches for ER classification. To overcome this limitation, AL has been proposed to select the most valuable data to be labeled to achieve reasonable effectiveness with less labeled data [Melville and Mooney, 2004]. As introduced in Chapter 2, state-of-the-art AL approaches can be divided into single-model-based and committee-based AL. Our research is related to the later one, also called Query By Committee (QBC). Based on our observation, applying QBC approaches for ER faces two challenges:

Diversified Committee: The key challenge to make QBC work in common is to generate a diversified committee, which can lead to insightful voting disagreements so that the informativeness of data can be represented and distinguished [Lu et al., 2009; Melville and Mooney, 2004]. To achieve diversity, i.e., generating various

voting results from the committee, for the vast majority of proposed AL approaches, they consider how to get multiple models with only one single type of classification algorithm. So far, several methods have been proposed for the diversity purpose, such as query by bagging, query by boosting [Mamitsuka et al., 1998]. However, for all those ensemble-based approaches, the accuracy of each model is compromised to get this diversity. For instance, in the bagging approach, the initial training dataset is divided into several smaller subsets, then different models are built based on different subsets [Mamitsuka et al., 1998]. Those trained models cannot be expected to achieve such accuracy as the model trained on the whole training dataset. Besides, nowadays, data is also quite variable in their types and there is no universally best model for all types of data. If a system completely relies on a single type of model, accuracy could not be acceptable for the worst cases.

Imbalanced ER Classification: The second challenge specialized for an AL-ER solution is the generation of the initial training dataset. The binary classification task for ER is a special task because of the imbalance of its two groups. In our real world, there are much fewer matching pairs than non-matching pairs, e.g., for the well-known DBLP-ACM dataset, the imbalance ratio is 1 match but 1785 non-matches [Wang et al., 2015]. If the initial training dataset is randomly selected from all candidate pairs, the possibility to contain matching pairs would be quite low, which may lead to a very low starting accuracy of trained models or even fail in training a model. Facing imbalanced data, oversampling and undersampling are commonly-used. However, except for their intrinsic shortcomings (overfitting for oversampling and discarding potentially useful data for undersampling [Kotsiantis et al., 2006]), they also contradict the goal of AL: saving labeling effort as much as possible.

Facing both challenges, in this chapter we propose a novel **He**terogeneous **A**ctive **L**earning **E**ntity **R**esolution (HeALER) solution. We specifically detail our contributions as follow:

- We design a specialized technique to generate the initial training dataset, which is suitable for the inherent class imbalance in ER;
- We propose to construct the AL committee with different types of classification algorithms, through which we can achieve diversity, accuracy and robustness requirements of a committee;
- We prototype our solution and evaluate it with two well-known ER benchmarking datasets, and compare it to passive ML and two state-of-the-art AL-ER approaches (ATLAS [Tejada et al., 2001] and ALIAS [Sarawagi and Bhamidipaty, 2002])[1]. The evaluation results show that HeALER is faster to converge and can reach a higher final F-measure, which also indicates that with fewer labels a satisfactory F-measure can be achieved.

The remainder of this chapter is organized as follows: In Section 7.1, we introduce our HeALER approach. Subsequently, we evaluate our approach and discuss the

[1]The source code and the datasets can be found at https://git.iti.cs.ovgu.de/Chen/entity-resolution-for-big-data.

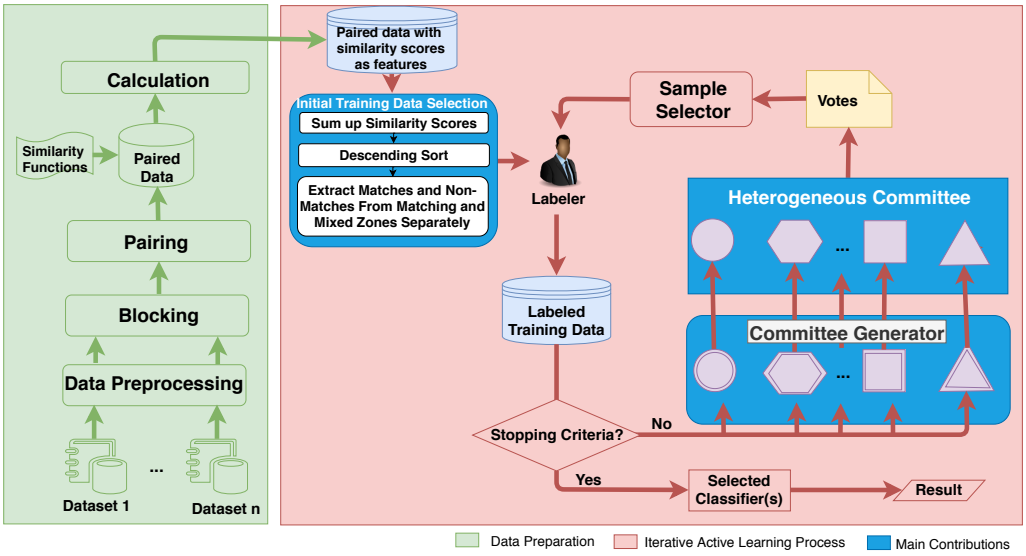


Figure 7.1: The global workflow of HeALER.

experiment results in Section 7.2. Before we summarize this section in Section 7.4, we also compare our method to other related work in Section 7.3.

7.1 Our HeALER Method

In this section, we introduce our designed QBC AL method for ER, which is characterized by its initial training data selection approach and its heterogeneous committee. We start with a global picture of our approach in Section 7.1.1, then we represent our initial training data selection method and the heterogeneous committee in the following sections.

7.1.1 The Global Workflow

Figure 7.1 represents the global workflow of our method HeALER. It is separated into two parts, the left green area describing the preparation steps, and to the side the light red area corresponds to the AL stage.

Preparation for Active Learning As we can see from the left green area of Figure 7.1, several standard preparation steps are required to start the ER process. At first, input data is preprocessed if necessary, which may include data cleaning, formatting, standardization. Afterward, blocking is performed to omit unnecessary comparisons, which are obvious non-matches based on predefined blocking keys [Chen et al., 2018b]. Then candidate pairs are generated based on the blocking result. Subsequently, for each attribute pair, one or more similarity functions are chosen to best calculate similarities between each attribute pair to get similarity scores as features for the following learning-based classification step [Chen et al., 2019a]. For the above-introduced steps, proper techniques should be employed based on ER task requirements, our contributions are reflected in the AL part, which will be briefly introduced next.

Iterative Active Learning Process The first step of HeALER is to select pairs from the candidate pairs to be labeled by domain experts for an initial training

dataset. As mentioned in Chapter 1, the classification step of ER is an imbalanced binary classification problem, i.e., there are much fewer matching pairs than non-matching pairs [Elmagarmid et al., 2007]. To reach a relatively high starting point with the initial training dataset, the training data is required to be balanced and informative. Balanced means the initial training dataset should contain sufficient percentages of matching and non-matching pairs, which is hard to achieve when one randomly picks pairs from the entire input data, since a too high percentage of non-matches would be selected. Informative means the initial training data could involve useful information, which can benefit classifiers. The details of how we achieve both goals will be introduced in Section 7.1.2.

Based on the initial training dataset, different classifiers are trained on them and then all classifiers together form the required committee. Notably, our classifiers are trained by different classification algorithms, which means our committee is heterogeneous. Compared to the majority of state-of-the-art QBC-AL approaches, our heterogeneous committee has the following advantages: First, the fundamental requirement - diversity of the committee - is achieved in a natural way without any other efforts. Second, each member of the committee is trained with the best or full ability without any compromise, which is more promising to provide a more accurate vote. Last, the committee analyzes training data and provides the result from multiple perspectives, no matter which kind of data the committee is facing, it can provide relatively stable and acceptable results. The methods to form our committee, including how to define the number of required classifiers and how to select classification algorithms as committee members, will be explained in Section 7.1.3.

After the committee is formed, they are employed to vote each pair from the unlabeled pool into match or non-match. The calculation of the disagreement of voting results for pairs will be firstly represented in Section 7.1.4. Then this process is iterated until the stopping criteria are reached.

7.1.2 Initial Training Dataset Generation

As explained in the last section, a good initial training dataset should be balanced and informative. To achieve both criteria, we analyzed a learning-based ER process. The resources that we have for the classification step are the candidate pairs and already calculated similarity scores for each attribute pair as features. Figure 7.2 is a histogram formed for the benchmarking bibliography dataset ACM-DBLP [Rahm, 2017], which describes how the percentages of matching and non-matching pairs vary along with different similarity score levels. There are four attributes in this dataset. From these, a total of 16 similarity scores are calculated as features (five similarity scores for attributes *title, author, venue*, each with five string similarity calculation functions; and one similarity score calculated for the attribute: publication year). Each separate similarity score is normalized between zero and one, and then the total similarity scores of all pairs should be between zero and sixteen by summing up all similarity scores. Based on this, we divided all candidate pairs into 15 groups and each group is an interval between n and $n+1$ (n is from 0 to 15). As we can see from it, globally the whole pairs are located in three zones. For areas with the lowest similarity scores, the vast majority of pairs are non-matching pairs (the non-matching zone). Then the percentage of matching pairs increases in relatively middle

levels (the mixed zone), and for the last levels with the highest similarity scores, the vast majority of pairs become matching pairs (the matching zone). Dealing with variable datasets, the concrete ranges of the three zones may vary, however, globally speaking, those three zones and their trends should be valid for almost all datasets.

From the perspective of balance, the difficulty for the imbalanced classification step of ER is to find a sufficient number of matching pairs, while non-matching pairs are quite easy to get because there are much more non-matching pairs than matching pairs in our real world. The percentages shown in the figure can indicate the difficulty to get matching and non-matching pairs. To get sufficient matching pairs, the matching zone has to be focused. To get sufficient non-matching pairs, both the non-matching zone and the mixed zone can be the candidates.

From the other perspective of being informative, those pairs that are intrinsically difficult to classify based on available features can be considered as informative data since the classifier would be significantly improved if informative pairs are labeled and added to help the classifier training. Hence, those error-prone pairs should be true matching pairs with relatively *low* similarity scores and true non-matching pairs with relatively *high* similarity scores. True matching pairs with relatively *low* similarity scores should be located in the mixed zone, but it is not possible to get them, since the matching pairs account for very small percentages in the mixed zone. Therefore, for achieving both balance and informativeness, we have to pick matching pairs from the matching zone. On the other hand, for non-matching pairs, those that have relatively *high* similarity scores are more informative and locate in the mixed zone, by combining the conclusion from above, i.e., non-matching pairs can be obtained from the non-matching zone and the mixed zone from the perspective of balance, the mixed zone is the aiming zone for high-quality non-matching pairs.

Based on the above considerations, we conclude our method to generate the initial training dataset for learning-based ER in the following way:

1. First, there can be many similarity scores calculated and for different attributes, values of similarity scores may vary much. Hence, it is difficult to look into each separate similarity score and judge the possibility based on them separately. Therefore, we calculate a total score of each pair by summing up all similarity scores of attributes.

2. Next, we sort all candidate pairs based on their total scores in descending order.

3. Last, we divide all sorted pairs into k groups, then we can get the initial training dataset by randomly picking $n/2$ pairs from the top k_1 groups (the matching zone) for getting sufficient matching pairs and $n/2$ pairs from the next k_2 groups (the mixed zone) for getting sufficient and informative non-matching pairs (n is the preset number of initial training data). There is no accurate method to determine which k, k_1, k_2 are the best. The following hypotheses can be used. If the ER problem is between two data sources and the linkage is one-to-one linkage, the highest number of matches is the number of records in the smaller dataset. This number can be used as the size of the

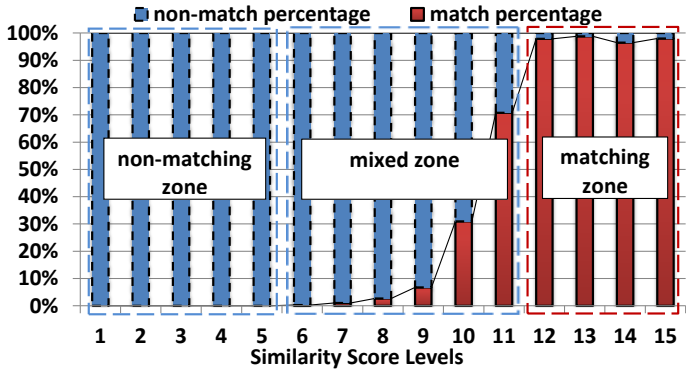


Figure 7.2: Distribution of similarity scores.

matching and mixed zones. If the linkage is one to many, even many to many linkages, the information that can be used is the approximate percentage of matching pairs, then this can be the basis to locate the matching zone and the same percentage of pairs can be counted for the mixed zone. If the percentage of matching pairs is unknown, as a rule of thumb, 10 groups should be a good number to averagely divide all pairs with a proper blocking step. Then the matching zone is the top group with the highest similarity scores and the mixed zone corresponds to the second group for getting non-matching pairs.

With the above-introduced strategy, the interesting areas analyzed above are established. With the first top groups, we are able to get sufficient matching pairs, and with the next groups, sufficient and informative non-matching pairs can be obtained.

7.1.3 Heterogeneous Committee

As introduced above, our committee is heterogeneous, which means that the classifiers of the committee are trained with different classification algorithms. The method designation focuses on two aspects: how many classifiers and which classifiers to choose.

Generally speaking, our heterogeneous committee is allowed to contain any number of classifiers. Based on the result in [Sarawagi and Bhamidipaty, 2002], the performance of the classifier is not too sensitive to how many members a committee has and with four classifiers the aggregated accuracy is already satisfactory enough. On the one hand, each additional member in the committee means one more training process per iteration, which can heavily increase the time needed for generating one round committee and has a negative impact on efficiency. On the other hand, having more than four members for the committee achieves even lower accuracy [Sarawagi and Bhamidipaty, 2002]. Therefore, in our evaluation, four classifiers are generated to form the committee. Next, we present which candidate algorithms are suitable to be committee members. In general, we considered the following factors:

Accuracy with little training data: The selected classifiers should have relatively high accuracy. Particularly, because the purpose of using AL is to reduce required human labeling efforts, we assume that for the training dataset, not so much training data is required to achieve high accuracy, which means that the selected classifiers

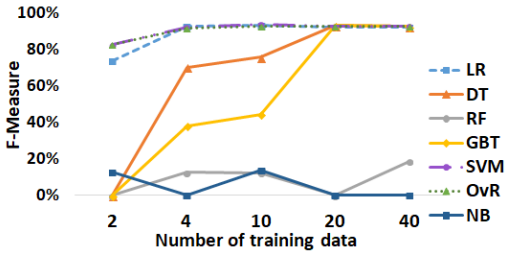


Figure 7.3: F-measure comparison of seven classification algorithms.

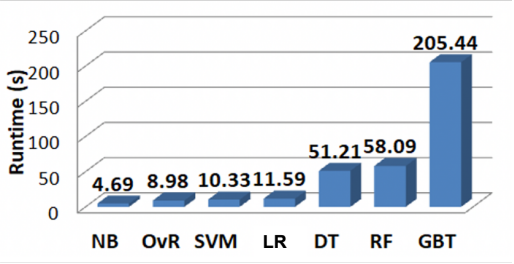


Figure 7.4: Efficiency comparison of seven classification algorithms.

should still work when only little training data is available. This is the main factor we use to choose classification algorithms.

Efficiency: Efficiency also requires consideration, since a learning-based classification is much more time-consuming than a simple threshold-based classification and such factor can be expected to have a large impact on the performance, as data grows.

Interpretability: Interpretability is also of great importance for choosing the learning algorithms because we can use machine learning responsibly to ensure that our values are aligned and our knowledge is reflected [Doshi-Velez and Kim, 2017].

We considered the following seven common binary classification algorithms: logistic regression (LR), decision tree (DT), random forest (RF), gradient-boosted tree (GBT), support vector machine (SVM), one-vs-rest logistic regression (OvR), and naive Bayes (NB). To select classifiers for our committee, we evaluated their F-measures also on the same benchmarking bibliography dataset ACM-DBLP [Rahm, 2017] used in Figure 7.2 by using different sizes of initial training data. Figure 7.3 shows the results. SVM, OvR, and LR have a satisfactory F-measure value even with only two training data pairs. NB and RF provide still a very low F-measure value even with 40 training data instances. NB classifiers are generative models that need prior probabilities. The probabilities are inaccurate for our case because our initial training data is chosen by our proposed method in the last section, which normally generates relatively balanced training data. This state of training data does not conform to the test data [Nguyen and Smeulders, 2004]. Besides, it assumes that all features are independent [Rennie et al., 2003]. However, our features are actually not independent, which may lead to the low F-measure for NB classifiers. The RF classifier cannot perform well, because it trains an ensemble of decision tree classifiers by splitting the training dataset into multiple subsets, then chooses subsets of features for each decision tree classifier [Meng et al., 2015]. This leads to a low F-measure especially when there is not enough training data. DT overall performs well except for the case with two training data pairs, in which the DT classifier is not possible to be trained. GBT is in a similar situation as DT. However, its F-measure values are always lower than DT. We also evaluate the efficiency of all seven classification algorithms. All candidate pairs are divided into roughly two equal groups. Training data is generated by randomly picking four matching and non-matching pairs from the first group and then test data is the other entire group. The results are shown in Figure 7.4. As we can see from it, results show that DT, GBT, and

RF need obviously more time than the other algorithms. NB runs the fastest, OvR, SVM and LR follows. However, all three tree-based classification algorithms DT, RF, and GBT are quite slow. By combining the perspectives of interpretability and efficiency with the accuracy result, SVM, OvR, LR, and DT are selected to form our heterogeneous committee.

Above we provided guidelines on how to choose classification algorithms to form the heterogeneous committee. Facing different implementations of algorithms with different adopted libraries, the best choices of classification algorithms may change case by case.

7.1.4 Training Data Candidate Selection and Termination Conditions

After our heterogeneous committee is formed based on the above-introduced approach, it is used to vote unlabeled pairs as matches or non-matches. Then those pairs with the most disagreement are those interesting pairs that we may select to be labeled by domain experts and added to the training dataset. The disagreement value of the voting results for pairs is calculated with the following equation:

$$Disagreement(pair) = \sum_{(a_m,a_n)\in committee)} Difference(result(a_m), result(a_n)) \quad (7.1)$$

where (a_m,a_n) are the combinations of results from any two classification algorithms from the committee and the *Difference(x, y)* function returns zero or one, depending on whether x equals to y or not. With this equation, we sum up all the differences between any two combinations of classification algorithms as the final disagreement value of the votes. However, the pair with a high disagreement value has also a high possibility that it is an outlier. If an outlier is selected and added to the training dataset, it will negatively impact the performance of classifiers. To reduce the possibility that outliers are selected, the random sampling proposed in [Sarawagi and Bhamidipaty, 2002] randomly picks the pair from the *top-n* pairs to alleviate the probability that an outlier is selected to be labeled, n can be set manually, such as 10, 20, 30.

Then the training data is updated in the above-introduced way iteratively till a preset termination rule is reached. The termination rules can be considered from the following possibilities:

- *Iteration rounds:* As an iteration process, the most straightforward way is to stop after a preset number of iteration rounds. It can be set by experience when the rough round of its converge can be estimated.
- *Running time:* In reality, the iteration process can also be ended manually, when the acceptable processing time runs out.
- *Training data size:* The domain experts can also stop labeling when they are not able to label more data. Then the capability of how much data they can label is also another possibility to terminate the process.

Table 7.1: Datasets used in experiments.

Datasets	#Input	#Records in DBLP	#Records in ACM/ Scholar	#Pairs (#matching pairs)	#For training data selection	#For testing
ACM-DBLP2	2	2616	2294	21095 (2189)	10547	10548
Scholar-DBLP1	2	2616	64263	44999 (4351)	22500	22499

- *Predicted accuracy:* It is often common that the users of the active learning ER system have a requirement for precision, recall, or F-measure values. In this case, more extra data needs to be labeled as testing data to evaluate the current classifiers after each iteration. However, with this stopping condition, human efforts have to be increased, since a ground-truth for the test data has to be provided.

After the iteration process is completed according to preset termination conditions, the committee or a specific classifier can be used to identify duplicates for any unlabeled data.

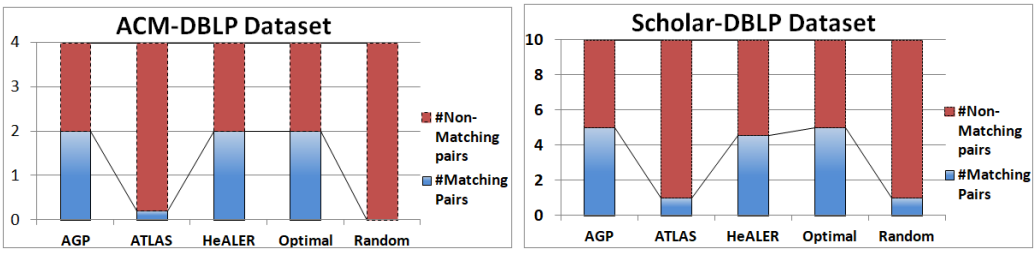
7.2 Evaluation

In this section, we evaluate HeALER from three aspects: first, we solely conduct experiments to evaluate the balance and accuracy of our initial training data selection method (Section 7.2.2). Second, we evaluate our heterogeneous committee and compare it to passive learning and committees formed by ALIAS and ATLAS (Section 7.2.3). Last, we evaluate our entire HeALER approach against an ML process and two state-of-the-art QBC-AL approaches: ALIAS and ATLAS (Section 7.2.4). For all results, the accuracy is measured using F-measure.

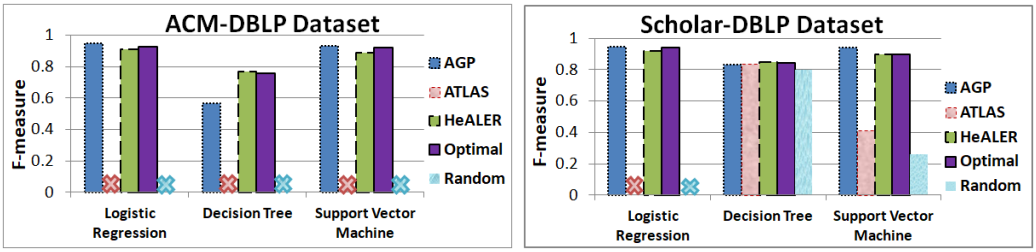
7.2.1 Experimental Setting

Datasets: We evaluate HeALER on two commonly-used real-world datasets: ACM-DBLP and Scholar-DBLP citation datasets [Rahm, 2017]. Both datasets include two parts, one part is from the DBLP citation database and the other one is from ACM or google scholar citation databases, respectively. All of them have four attributes, including title, authors, venue, and publication year. To prepare data for HeALER, we have done the following steps based on the two original citation databases: We first preprocess both databases by removing stop words and null values. Then we generate blocking keys (the first five letters of the title) for each record. Subsequently, we join two database tables with the blocking key as the join attribute, so that we get all candidate pairs. Afterward, similarity functions are performed on each attribute to get corresponding features. For attributes "title", "author", we apply cosine, Jaccard, Jaro-Winkler, metric longest common subsequence, N-Gram, normalized Levenshtein, and Sorensen-Dice similarity functions [2]. For attribute

[2]Implemented by the Debatty library (version 1.1.0).



(a) Balance of matching and non-matching pairs.



(b) F-measure comparison.

Figure 7.5: Evaluation of initial training dataset selection approaches.

"venue", the Jaccard similarity function is used. For the last attribute "year", the similarity between two values is one or zero based on whether they equal or not. In this way, we obtained 16 features. For the preparation of our initial training dataset selection method, total similarity scores are calculated and appended to data as well. With the above-introduced steps, for the ACM-DBLP dataset, we got 21095 pairs after blocking (including 2189 true matching pairs). We randomly divide all pairs into two parts: the first half 10547 pairs as the first part form the dataset to select training data and the remaining pairs for testing. For the Scholar-DBLP dataset, we got 44999 pairs after blocking (including 4351 true matching pairs). We also randomly separate it into two parts in the same way as the ACM-DBLP dataset. The details of the datasets are summarized in Table 7.1.

Implementation Related: Since learning-based classification is much more time-consuming than threshold-based classification, we implemented HeALER with Apache Spark (version 2.4), as preparation for big data processing. However, in this thesis, we focus only on the quality side of ER results. The classification algorithms used are implemented with Spark MLlib. The programming language is Scala with the version 2.11.12.

7.2.2 Initial Training Dataset Evaluation

Experimental Design. This experiment is to evaluate different strategies to select the initial training dataset by getting the average results over five runs. We use both datasets in Table 7.1. The tested initial dataset sizes are four and ten, which are proved to be the least to function selected classifiers (see Figure 7.3). The following strategies are evaluated:

Random Selection: It means we randomly select the required number of pairs.

Optimal selection: The optimal selection means that training data is optimally balanced, i.e., because we have the ground truth for our datasets, we pick half

matches and half non-matches from the unlabeled data. However, this is not practical, since, before labeling, we have no idea which pairs are matches or non-matches. In [Sarawagi and Bhamidipaty, 2002], they selected initial training data in this unpractical way.

Initial training data selection of ATLAS [Tejada et al., 2001]: ATLAS ranks all pairs on their total similarity scores, then divides the whole pool to n groups (4 or 10 groups for two tested dataset sizes respectively), at last, the initial training dataset is obtained by randomly selecting one data pair from each group.

Initial training data selection of AGP [de Freitas et al., 2010]: To get both matching pairs and non-matching pairs, the initial training dataset of AGP is obtained by selecting half number of pairs with highest total similarity scores (2 or 5 pairs for two tested dataset sizes respectively) and the other half number of pairs with lowest total similarity scores (2 or 5 pairs for two tested dataset sizes respectively).

Initial training data selection of HeALER: Our own method HeALER selects the initial training dataset in the way of the hypotheses described in Section 7.1.2. Since the linkage for the ACM-DBLP dataset is one-to-one linkage, the highest number of matches is the number of records in the smaller dataset, i.e., 2294 records from the ACM library. As the whole dataset is almost equally split into two datasets. Then the matches contained in the first dataset to select training data should be 1147. This number can be used to get the matching and mixed zones, i.e., two pairs randomly picked from the first 1147 pairs with the highest similarity scores, and two pairs randomly picked from the next 1147 pairs. Regarding the other dataset Scholar-ACM, it is not one-to-one linkage, but we know that the approximate percentage of its matching pairs is 10, therefore, we divide all pairs into 10 groups, and the first top group with the highest total similarity scores is the matching zone, where we randomly get 5 pairs, and the second group is the mixed zone, where we randomly get the rest 5 pairs.

We evaluate those above-introduced selection methods with balance and F-measure metrics. For the balance metric, we show how many matching and non-matching pairs in the training dataset. For the F-measure metric, F-measures values are calculated by testing the classifiers trained on different training datasets with LG, DT, SVM classification algorithms respectively on the test dataset.

Results and Discussion. As can be seen from Figure 7.5a, with random and ATLAS approaches, the training data selected is quite skewed, no sufficient matching pairs are picked, especially the random selection for the ACM-DBLP dataset selects no matching pairs, which may make the training data unusable since some classifier algorithms cannot work with only one class of data for a binary classification problem. HeALER can achieve relatively balanced training data, but not as completely balanced as AGP and Optimal selection. The F-measures using LR, DT, and SVM calculated on the training data selected with different approaches are shown in Figure 7.5b. Therein, the training data selected using ATLAS and the random approach works only for DT and SVM on the Scholar-DBLP dataset. For all other cases, no classifiers are successfully trained and used for the later test classification because of exceedingly skewed training data. The other three approaches work apparently better. With the training data they selected, it is always possible to complete the

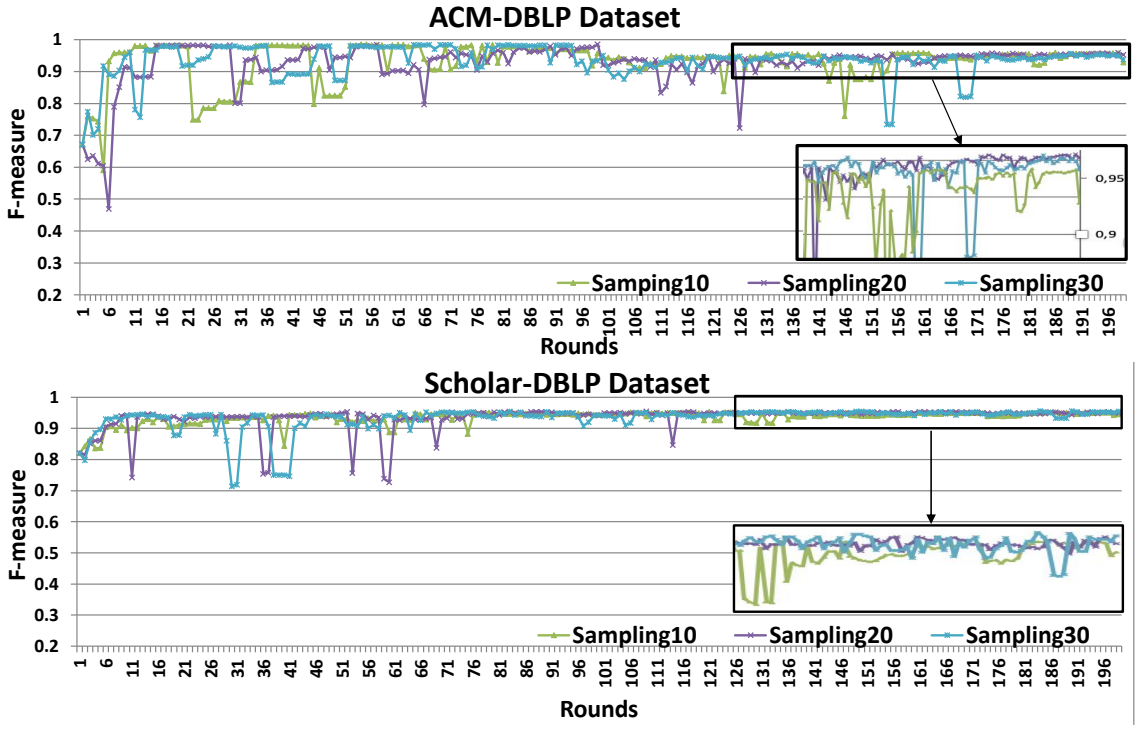


Figure 7.6: Comparison of different sampling strategies.

classification tasks using the trained classifiers. Particularly, HeALER outperforms AGP and the optimal case with DT due to the more informative training data, which makes the splitting closer to the truth. However, it achieves a bit lower F-measure for LR and SVM. By concluding the results, we can say that the quality of HeALER training data is high when the number of divided groups can be correctly defined. Otherwise, the AGP strategy can be applied to achieve acceptable F-measure.

7.2.3 Heterogeneous-Committee Evaluation

Experimental Design. This experiment is designed to specially evaluate our heterogeneous committee and compare it to other approaches (committees formed in [Tejada et al., 2001] and [Sarawagi and Bhamidipaty, 2002] and passive learning to randomly pick pairs without basing on committees' decisions). Both datasets in Table 7.1 are used. We fix the initial training data selected by our own strategy for all approaches, which provides them fair and good starting points. To determine how many top pairs are used for random sampling, we ran experiments to compare sampling strategies with 10, 20, and 30 top pairs with our HeALER method on both datasets. Figure 7.6 shows the comparison results. As we can see from them, for both datasets, the Sampling20 approach performs the best, achieves higher F-measure also at a faster speed. Therefore, the Sampling20 strategy is used as the countermeasure to reducer the selection of outliers. After each iteration of the AL process, the F-measure is calculated on the classification results of the test data in Table 7.1 obtained by using the DT classifier trained on the updated training datasets by each approach. The AL process terminates after 199 rounds. Each experiment is repeated three times to get the final average result. The details how different approaches perform are introduced as follows:

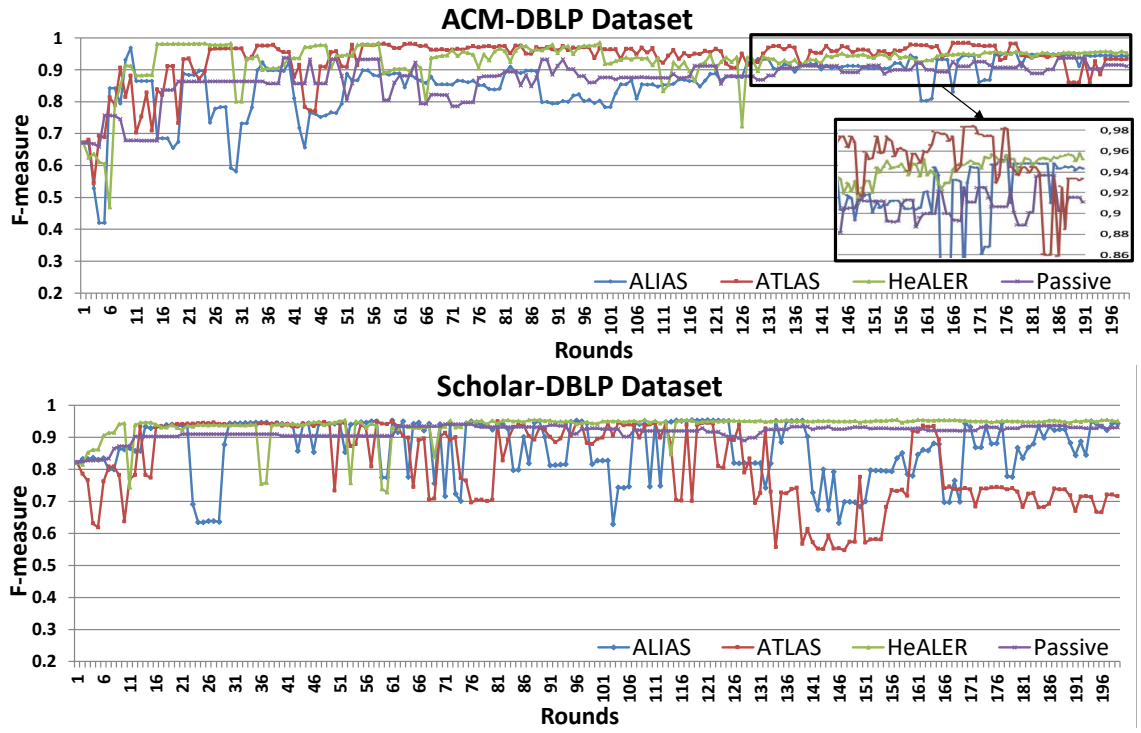


Figure 7.7: Comparison of different committees.

Passive Learning: This approach randomly picks pairs to be labeled by humans and added to the training dataset without relying on any committee votings.

ALIAS Committee [Sarawagi and Bhamidipaty, 2002]: ALIAS forms its committee by randomizing parameters while training classifiers with the selected algorithm. In our experiments for both datasets, the SVM algorithm is used. We vary the value of its parameter for the maximum number of iterations with 4, 6, 8, and 10. Then four classifiers are trained respectively and form its committee. In the ALIAS paper, for their experiments, they applied the DT algorithm and varied the parameter where to split. However, as our implementation depends on the Spark MLlib, it is not possible to adjust this parameter. Therefore, we apply the SVM algorithm for our experiments.

ATLAS Committee [Tejada et al., 2001]: ATLAS partitions the training dataset to four subsets, then each subset of the training data is used to train its classifier to form its committee. The classification algorithm used here is the same as ALIAS: SVM for the purpose of comparison. For both datasets, each time 80 percent of pairs are randomly chosen to constitute the training dataset. Four subsets are required to get four classifiers of the committee.

HeALER Committee: As explained in Section 7.1.3, our heterogenous committee includes four classifiers, which are trained with SVM, OvR, LR, and DT algorithms, using the complete training dataset.

Results and Discussion. Figure 7.7 shows the comparison results of different committees and passive learning. As we can see from the results of ACM-DBLP dataset, the F-measures of all approaches fluctuate much in the first 50 rounds, then becomes more stable later on. After about 140 rounds, our heterogeneous committee

keeps F-measures higher than 0.92 and reaches its rough convergence. In contrast, ALIAS and ATLAS committees still cannot achieve their convergences till 199 runs. They show even less stable and lower results than passive learning. However, the highest F-measures they are able to reach during the experiments are much higher than passive learning, which proves the effectiveness of the committee to explore which are more informative pairs. Since passive learning randomly chooses more pairs to be labeled, the informative pairs are hard to be selected to really cover the shortages of the classifiers. From the result of the other Scholar-DBLP dataset, we can get similar conclusions. Our heterogeneous committee converges already after about 110 rounds and keeps the F-measure 0.95 afterward. ALIAS and ATLAS committees are far from their convergence even with 199 rounds. Passive learning works quite well for this dataset due to the high initial F-measure. However, it requires much labeling effort to improves its F-measure. To summarize the results, our heterogeneous committee shows its advantage in picking informative data to improve the F-measure of the classifier and reach convergence with much less labeling efforts than passive learning, ALIAS, and ATLAS committees.

7.2.4 Overall Evaluation and Comparison

Experimental Design. After we evaluate our initial training data selection approach and our heterogenous committee separately, in this section, we evaluate our entire HeALER approach by comparing the F-measures using a one-off ML approach, ALIAS and ATLAS approaches based on the same number of training data. For this overall evaluation, the same datasets are used as in the last two sections. The F-measures are all calculated on the classification results of the test data in Table 7.1 obtained by using the DT classifier. All approaches except the normal machine learning approach follow the iteration process of AL, and terminate after 199 rounds. For the normal one-off ML approach, we randomly picked the corresponding number of training data of each iteration and calculate the F-measure of the test data using the DT classifier. ATLAS has no strategy to reduce the possibility to get outliers but it chooses the pair with the highest similarity value among all pairs with the highest disagreement value. Therefore, in the overall evaluation, for ATLAS, this approach choosing the pair with the highest similarity value is used. For HeALER and ALIAS, the sampling20 strategy is used as in the committee comparison experiment. The final result is averaged by three times' repetition.

Results and Discussion. Figure 7.8 shows the comparison results of different AL approaches and a normal ML process. As we can see from the results of the ACM-DBLP dataset, HeALER has the highest initial F-measure and keeps an F-measure around 0.9 with 20 or more training data. ALIAS and ML perform the worst and fluctuate their F-measures from the beginning to the end. ML starts to function stably with at least 33 labeled data and cannot significantly improve its F-measure when labeling more data. ATLAS starts to work with 10 labeled data and hardly varies its F-measure. The reason can be because its strategy always selects data with the highest total similarity score, weakens the effects of the disagreement values of data, and often chooses the same data for different iterative rounds, which leads to changeless F-measure for several or even dozens of iterative rounds. Although it seems that ATLAS performs quite well, the results of the Scholar-DBLP dataset, in which ATLAS performs the worst, shows that ATLAS is not reliable, more research

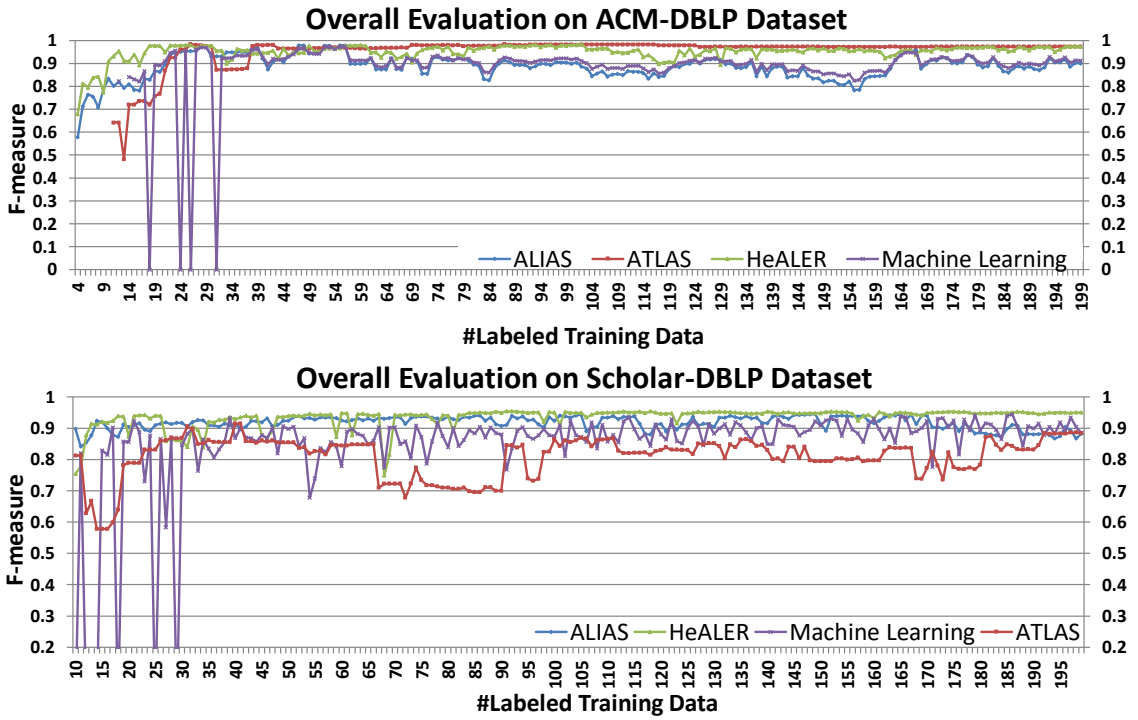


Figure 7.8: Overall evaluation of different active learning approaches.

on the strategy of selecting the highest similarity score from the data with the highest disagreement values is required. For the results of ALIAS, HeALER, and ML on the Scholar-DBLP dataset, similar conclusions can be made. The results show that HeALER works better than the other compared approaches.

7.3 Related Work

We now discuss the existing AL approaches to reduce the labeling efforts for learning-based ER. The AL related approaches for ER include similar approaches as the common AL with a goal of selecting the most informative data for classifiers to be labeled by humans (single-model-based [Ngomo et al., 2011], committee-based [de Freitas et al., 2010; Isele and Bizer, 2013; Ngomo and Lyko, 2012; Ngomo et al., 2013; Sarawagi and Bhamidipaty, 2002; Tejada et al., 2001]), and special AL approaches as well for the purpose of getting the best rules (like classifiers) that are able to provide high precision without considering the quality of training data [Arasu et al., 2010; Bellare et al., 2012, 2013; Fisher et al., 2016; Qian et al., 2017]. Therein, Ngomo et al. [Ngomo et al., 2011] identify the most informative data to be labeled and added into the training dataset with the maximized convergence of the used classifier. The proposed committee-based AL approaches differ from each other globally with different committee forming approaches. The approaches [de Freitas et al., 2010; Isele and Bizer, 2013; Ngomo and Lyko, 2012; Ngomo et al., 2013] use genetic programming algorithms to learn multi-attribute functions. However, the quality of those functions cannot be guaranteed.

The research approaches [Sarawagi and Bhamidipaty, 2002; Tejada et al., 2001] are the most similar to ours. They form their committees with several classifiers, which

are trained on a single type of classification algorithm. However, to achieve diversity of classifiers in the committee to make AL work with the most disagreement strategy, their classifier qualities are compromised, which restricts the ability of the committee to identify the most informative data. Moreover, the initial training dataset selection problem is not correctly handled. Sarawagi and Bhamidipaty [Sarawagi and Bhamidipaty, 2002] directly assume that the AL process starts with an initial training dataset including five matching and non-matching pairs, which is not realistic since it cannot be known whether a pair is matching or non-matching before labeling. Although in the other paper [Tejada et al., 2001], this reality is considered, however, the initial training dataset they selected is quite biased with the number of matching and non-matching pairs, which leads to very low quality of classifiers for the beginning iterations. In contrast to them, our proposed HeALER can provide a high-qualified initial training dataset and the heterogeneous committee can select more informative data to improve the classifiers faster.

7.4 Summary

In this section, we propose our AL approach HeALER for ER, which could select a relatively balanced and informative initial training dataset and use its heterogeneous committee to select informative pairs to be labeled by the human to improve the classifier. We evaluated and compared it with the passive (machine) learning and two state-of-the-art AL-ER approaches ATLAS and ALIAS. The evaluation results show that HeALER is faster to converge and can reach a higher final F-measure than other approaches. In addition, the results also indicate that it requires less training data to reach a satisfactory F-measure, which conforms to the purpose of using AL approaches: reducing human labeling effort. However, we also observed the fluctuations during the early rounds, which are caused by choosing outliers to the training dataset.

8. Conclusion and Future Work

Although the research on Entity Resolution (ER) has already a long history, nowadays it still keeps active and attractive. Facing the challenges big data brings to it, the research potentials are further extended and corresponding solutions are called. This thesis responds to this need and makes contributions to make ER more efficient and effective for high-volume and variable data. In the following, we conclude our contributions in each challenge and briefly discuss potential future work.

8.1 Conclusion

To deal with **high-volume data**, ER solutions need to be more efficient and scalable. In our thesis, Chapter 3, Chapter 4, and Chapter 5 presented our contributions in this aspect and focus on solutions using parallel computation for ER, which are concluded in the following based on the research questions raised in Chapter 1:

- To answer RQ1: *"What is the state-of-the-art and the latent research directions of parallel ER?"*, in Chapter 3, we conducted a Systematic Literature Review (SLR) on the area of parallel ER and overviewed a corpus of 58 articles. According to three groups of extracted criteria: general, effectiveness-based, and efficiency-based, we represented the state-of-the-art research and summarized them in multiple tables, which can be used for a quick look-up for existing research approaches of parallel ER. Furthermore, we discussed the reviewing results in each table and concluded possible solutions for each group of criteria, which can provide guidance for the designation of future parallel ER processes. At last, based on the observations and conclusions from our SLR, we suggested several research directions, where future research should pay more attention.

- To answer RQ2: *"How do different implementation options provided by big data processing frameworks affect the performance of ER processes?"*, in Chapter 4, we took three available APIs in Apache Spark: RDD, DataFrame, and Dataset as examples to explore their performance to implement parallel ER. A typical pair-wise ER process including two scenarios: with (Scenario 1) or without

(Scenario 2) evaluation is considered and implemented by these three APIs. Moreover, different persistence options have been designed and evaluated for three APIs to find their optimal persistence option. From the perspective of implementation procedures, the high-level APIs DataFrame API and Dataset API, especially the possibility of integrating SQL queries into the DataFrame implementation, provides more convenience for programmers. In contrast, the RDD-based API provides more flexibility for programmers to define their own classes and functions case by case. Then we compared their runtime by submitting them to our Spark cluster. Based on our results, we get the following conclusions: For parallel ER with evaluation (Scenario 1), DataFrame implementation is the most efficient one. However, its superiority is not due to the automatic optimization provided by Spark, but the advantages from its best persistence option. Without considering persistence for the ER process, the RDD-based implementation runs faster than the other two implementations but cannot get benefit from persistence with our Spark cluster. For parallel ER without evaluation (Scenario 2), the RDD-based implementation is superior to the other two implementations, which conforms to the result of Scenario 1 without consideration of any persistence option. For many cases in reality, the evaluation step is not required. Hence, an RDD-based implementation is preferred to reduce the required time to get ER results.

- To answer RQ3: "How well can the state-of-the-art block-splitting-based load balancing strategies solve the load imbalance problem for parallel ER and what can be improved?", in Chapter 5, we in-depth analyzed the state-of-the-art approaches BlockSplit and BlockSlicer and pointed out their advantages and disadvantages theoretically. Then we proposed two other strategies TLS and BOS, which aim to overcome the limitations. Afterward, we implemented and evaluated them with our Spark cluster. We found that all four strategies could fulfill the fundamental requirement: balancing the reducer load with the help of a greedy strategy for assigning blocks to reducers on the majority of datasets. Besides, we exposed that the total runtime of a block-splitting-based load balancing approach can be affected by the other two factors: the required overhead and the GC time when facing limited cluster resources. Under different situations, no strategy can always outperform other strategies and we concluded their corresponding performance case by case. However, our proposed strategies TLS and BOS can provide similar and in the vast majority of cases more efficient performance than BlockSplit and BlockSlicer due to the fact that they reduce the overhead by block-wise assigning composite keys.

To deal with current more **variable data**, in our thesis, we considered two folds of problems for ER.

- The first problem is, under the circumstance of using word embedding for calculating similarities for all types of record attributes of ER, the effectiveness may be negatively affected by the inaccurate similarities of non-semantic attributes. Therefore, we have our RQ4: *"Based on the availability of both traditional similarity functions and the word-embedding-based similarity calcu-*

lation approach, how to choose between them to provide ER the highest effectiveness?".

To answer it, in Chapter 6, we classified record attributes into three groups: numerical attributes, non-semantically related attributes, and semantically related attributes. We designed different combinations of similarity measures based on the three groups of attributes and evaluated them on real and synthetic datasets. According to our experiment results, we have the following observations: first, a word-embedding-based approach works predominately better for semantic attributes. For non-semantic attributes, it may also be possible to achieve an F-measure which is comparable or worse than traditional similarity measures. However, for numerical values, word embedding is not recommended since a hybrid approach shows obviously better F-measure than only using word embedding for the Amazon-Google dataset with XGBoost and RF classifiers. Therefore, in conclusion, the safest way for similarity calculations of ER problems is to use word embedding for semantic attributes and to use traditional similarity measures for non-semantic attributes.

- The second problem we concern is, due to the absence of training data, much human effort is required to label records to achieve satisfactory effectiveness of ER results. To alleviate this, active learning (AL) can be used. However, committee-based AL has not been sufficiently studied for ER. Based on the shortages of existing approaches, we have our RQ5: *"How to select a balanced and informative initial dataset and form an effective committee for committee-based active learning ER?"*.

  To address this, in Chapter 7, we proposed our own committee-based AL approach named HeALER, which is powered by its initial training data selection approach and heterogeneous committee. We implemented our HeALER approach, a normal machine learning-based ER, and two existing committee-based AL approaches: ALIAS and ATLAS, and evaluated them. Our experiment results showed that HeALER is superior to ALIAS and ATLAS in higher F-measure values and faster coverage speed and is able to reach satisfactory effectiveness with less labeled data than machine learning based ER and the other two AL approaches.

To summarize, our thesis concerns the challenges big data brings to ER and proposes solutions from different perspectives. The research in Chapter 3 and Chapter 4, and Chapter 5 contributes to efficiency and scalability by surveying the main parallel ER research, suggesting the suitable implementation API, and improving block-splitting-based load balancing strategies for parallel ER. The research in Chapter 6 and Chapter 7 comes down to the effectiveness issues by improving effectiveness through hybrid similarity measures for the pair-wise comparison and achieving required effectiveness with less labeled data for a learning-based classification. All above-summarized solutions contribute towards our vision introduced in Chapter 1: providing the most suitable ER solutions under different big data scenarios by a specialized and powerful framework, into which our contributions can be integrated. To approach this vision, future work introduced in the next section should be considered.

8.2 Future Work

In this section, we present future work, which can further improve ER approaches to solve the challenges big data brings to it.

The first group of future work extends our current research.

- **Extending the comparison study of different implementation options in big data processing frameworks**: There are different directions to extend our comparison study part. First, our current comparison study specialized on three APIs within the Apache Spark framework, which can be extended to cross-framework comparisons and explore the beneficial features from existing frameworks for our envisioned specialized framework for ER. Second, the current ER process adopts some typical techniques for its included steps. More commonly-used techniques can be added to assess the robustness of different implementation options. Last, the datasets we used for our finished comparison study are only regarding personal information. More types of datasets can be considered to see whether the same observations can be collected.

- **Extending the research on load balancing strategies:** During our evaluation for four block-splitting-based strategies, we found that with limited cluster resources, the required GC time may significantly increase and lower the efficiency. Because the threshold is determined based on the number of reducers, the performance of the strategies deeply relies on a suitable number of reducers. It is difficult to find the best number of reducers, which can balance the overhead time and GC time for optimal efficiency. Under this situation, a resource-bounded and reduce-partition-independent threshold determination approach can be explored to improve the usability of the strategies. Furthermore, in our evaluation, we found that the performance of different strategies may vary facing datasets with different properties. More broad evaluation can be conducted for future work. At last, the pair-based load balancing strategy is the other solution proposed. Whether the state-of-the-art pair-based approaches can be improved remains for future work.

- **Extending the hybrid similarity calculation approach:** We envision two core challenges: On the one hand, previous work [Köpcke and Rahm, 2008] has shown that thresholding, with a given blocking solution, improves the learning process; similarly, work in ER with embeddings [Ebraheem et al., 2018] shows good results without quantifying the precise impact of blocking and thresholding. Therefore, future work should consider this factor for our hybrid solutions. On the other hand, the search for the optimal balance between the similarity measures used becomes essential, as we show in our current study. Though some guidelines can be adopted in this task for specific cases, as we propose, future work is required to achieve a general method for combining the approaches. This may be achieved through the following ways. First, in our research, for the same dataset results are not completely agreed with different classifiers. This can be especially observed for the KNN classifier, in several cases, its results are opposite to the results obtained using the other two classifiers. Therefore, more classifiers should be explored to figure out the hidden

rules. Next, experiments on datasets, whose attributes can be relatively even divided into the three groups we presented, should be made to draw more definite general conclusions for combining the approaches.

- **Extending the HeALER approach:** There are also different directions to extend our HeALER approach. First, in the current research, we analyzed that in order to get sufficient match pairs and high-quality non-match pairs, the initial training data should be selected for match and non-match pairs separately. We suggested different hypotheses on the possible ways to determine the specific intervals for selecting match and non-match pairs. However, this cannot provide a guarantee for absolute balance and informativeness. Methods to approach more absolute balance and informativeness are expected. Second, the sampling strategy has been used to lower the possibility of selecting outliers for the AL processes. However, this cannot avoid the selection of outliers. For future work, techniques to exploit the local density to handle imbalanced data and recognize outliers [Nanopoulos et al., 2002] should be studied in order to improve HeALER and make it reach the convergence faster. Third, we have evaluated HeALER on two benchmarking publication datasets, which is not as challenging as another benchmarking dataset: Amazon-Google product datasets. Evaluations are required on this dataset to see whether HeALER can also work well for the more challenging dataset. At last, although our HeALER approach can reduce the required training data to achieve satisfactory effectiveness, human experts still need to spend time waiting for the end of each iteration to label the data. Approaches such as semi-supervised approaches can be combined with HeALER so that only the initial training data is required to be labeled by human experts at the beginning and afterward human experts are free from labeling for each round of iteration.

In our thesis, we touched only on the first two Vs of big data and considered the two Vs separately. The other group of future work is related to the combination of different Vs and the remaining two Vs: velocity and veracity.

- **ER for both data volume and data variety:** In this thesis, we contribute separately to the data volume and data variety parts. For the data volume challenge, we have the research goal to make approaches more efficient. For the data variety challenge, we aim to implement ER more effective and with less human labeling efforts. For future work, the research to study the interaction between efficiency and effectiveness under different big data scenarios can be carried on.

- **ER on data velocity:** Data velocity means the fast speed of generating new data. Facing this challenge, real-time techniques should be adopted to implement real-time ER. Compared to the traditional off-line ER, real-time ER has its own characteristics and even higher demand for efficiency. For new-coming records, to accelerate their integration into the resolved data, except for the similar steps of an off-line ER process, maintaining indexes for the resolved data and the way to store the resolved data should also be considered.

- **ER on data veracity:** Data veracity means the truthfulness and reliability of the data. With the development of the web, data veracity becomes more typical for the so-called web data. The ER process relying solely on the value of the data may not provide satisfactory results, evidence from additional sources should be utilized in the future work to counter the data veracity challenge.

A. Appendix

The appendix is organized as follows. In Section A.1, we introduce our method to select state-of-the-art approaches for our SLR in the parallel entity resolution (ER) area. In Section A.2, we represent some example codes for implementing the entity resolution process with three Spark APIs in Chapter 4. In Section A.3, we show the calculated imbalance ratio of the five introduce load balancing strategies in Section 5.3.2 with 112 and 224 reducers.

A.1 Methodology of Literature Search

In this section, we describe the process of finding and selecting the 58 papers on parallel ER. The process includes the following steps:

Identification of a starter set of articles: The first task in the literature search is to identify a starter set of articles (i.e., primary studies) having a strong focus on the considered field. If this set is too large, the effort to identify related papers will be unacceptable, while in the opposite case, if the starter set contains too few articles, some important articles may be missed. As ER has many synonyms, the most commonly used aliases were also considered as keywords in our searches, to achieve a starter set covering diverse entry points to sub-fields of research. On the other hand, ER-related research in computer science has been going on for almost six decades, so the number of articles is large. Therefore, it was mandatory to use keywords distinguishing serial and parallel ER to limit the size of the starter set. To make the search string more restrictive we added the term 'partition' to it because it is used in all papers that address parallel processing as a focus. Based on these considerations, we defined the following search string:

> *("entity resolution" OR "record linkage" OR "data matching" OR "deduplication" OR "duplication detection" OR "similarity join") AND ("parallel" OR "distributed") AND "partition".*

This search string was used on four popular literature databases: ACM Digital Library, IEEE Xplore, SpringerLink, and Scopus. The retrieved articles from workshops, conferences, and journals of computer science formed the starter set of 2582 articles, consisting of 873 articles from ACM Digital Library, 285 articles from IEEE Xplore, 992 articles from SpringerLink, and 432 articles from Scopus.

Applying inclusion and exclusion criteria on the starter set: The second step is to define inclusion and exclusion criteria and then apply them on papers in the starter set. Our inclusion and exclusion criteria are introduced in the following:

Inclusion Criteria:

IC01: Including articles, which are published between 2005 and December 2019. 2005 is chosen, because the vast majority of the research regarding parallel ER is published after this year and we intended to focus on the current state of research. December 2019 is the time we did the literature search.

IC02: Including articles, which are written in English.

IC03: Including research papers, in which the research ideas and solutions are originally from authors themselves.

IC04: Including articles, whose topics are generic ER (similarity join included) and using parallel computation techniques to solve ER problems.

Exclusion Criteria:

EC01: Removing overlapping articles between different literature databases.

EC02: Removing articles, which are summaries of workshops and conferences, posters, editorials, etc.

EC03: Removing articles, which are secondary studies and usually contain no technical contribution, such as literature reviews, surveys, comparative papers, tutorials, etc. However, the related secondary studies are considered for presenting the related work in the latter part of this chapter.

EC04: Removing articles, whose focus is on general data mining, data integration, data cleaning, data storage, data classification, similarity search, etc.

EC05: Removing articles, that only address specifics of ER in application domains, such as geospatial, forensic, networking, multimedia domains.

EC06: Removing articles, whose focus is not on using parallel DBs or big data processing frameworks to support parallelizing the ER process, but either to improve efficiency by using non-parallel algorithms, parallelism offered by GPUs for local tasks, focusing on solving privacy problems, or query-time/streaming ER.

EC07: Removing articles, which do not give evidence, such as references, proofs, experimental results, to support their claims.

EC08: Removing articles based on the number of citations indicating low impact or importance. Because the number of citations increases over time after publishing, it is not reasonable to have a single lower-bound value. Therefore, we removed articles

with a citation number less than two times the publication age in years before the current year of 2020, i.e., $2 * (2020 - publicationyear)$.

Using the above-defined criteria, we applied them on the start set with the following sub-steps:

sub-step 1: Apply criteria, which are independent of article contents, i.e., IC01, IC02, and IC03, with the help of the filter functions of each search engine of the literature databases. After this sub-step, 1840 articles remained.

sub-step 2: Apply the criteria (IC03-IC04 and EC02-EC06) on article titles to include or remove articles. After this substep, 268 articles remained for consideration.

sub-step 3: Apply the criteria (IC03-IC04 and EC02-EC06) on article abstract to include or remove articles. After applying the criteria, 102 articles are left. And lastly by applying the impact criteria EC07 and EC08, 49 articles are left. By removing overlapping articles, 43 articles are left.

sub-step 4: Apply the criteria (IC03-IC04 and EC01-EC07) to include or remove articles from the literature set by reading the entire paper. After applying these criteria on the whole paper, three articles are removed and 30 articles are left.

Supplementing the literature set by checking the references of articles: After we had the list of identified 30 articles from the second step, we checked references from or to those articles, and then decided whether to add the retrieved articles to our literature set or not based on all criteria described above. After this step, we found 28 more articles and had 58 articles in total.

A.2 Implementation Details for a Common Entity Resolution Process with Three Spark APIs

In this section, we show our three Spark-based implementations for each step of the used ER process. The implementations are represented in the order of RDD-based, Dataset-based, and DataFrame-based for each step. We omit the details of implementing specific functions and aim to emphasize the implementation differences between different APIs[1].

```
1 //RDD Implementation
2 SparkConf conf = new SparkConf().setAppName("EntityResolution");
3 JavaSparkContext sc = new JavaSparkContext(conf);
4
5 //Dataset and DataFrame Implementation
6 SparkSession sparkSQL = SparkSession.builder()
7 .appName("EntityResolution")
```

Listing A.1: Initializing Spark

```
1 //RDD Implementation
2 JavaRDD<String> input = sc.textFile(args[0], #partitions);
3
4 //Dataset Implementation
```

[1]The full code can be found at https://git.iti.cs.ovgu.de/Chen/entity-resolution-for-big-data.

```
Dataset<String> input = sparkSQL.read().textFile(args[0], #
    partitions);

//DataFrame Implementation
Dataset<Row> input = sparkSQL.read()
.option("header", "true")
.option("treatEmptyValuesAsNulls","true")
.csv(args[0]);
```

Listing A.2: Loading data into Spark

```
//RDD Implementation
JavaRDD<Record> preprocessedData = input.map(new Function<String,
    Record>()  {
  public Record call(String line) throws Exception {
    Record sd = null;
    //cleaning step implementation omitted
    sd = new Record(fields);
    return sd;
  }
});

//Dataset Implementation
Dataset<CsvRecord> preprocessedData = input.map((MapFunction<String
    , CsvRecord>) rec->{
    //Omitted, the same as RDD-based implementation
}, Encoders.bean(CsvRecord.class));

//DataFrame Implementation
Dataset<Row> preprocessedDataTem = data.na().fill("0");
Dataset<Row> preprocessedData = preprocessedDataTem.na().replace(
    cols, ImmutableMap.of(unwanted,""));
```

Listing A.3: Preprocessing

```
//RDD Implementation
JavaRDD<Record> withKeyData =  preprocessedData.map(new Function<
    Record, Record>() {
  public Record call(Record rec) throws Exception {
    DoubleMetaphone doublemetaphone = new DoubleMetaphone();
    String key = "",
    //blocking key definition omitted
    rec.setBlockingKey(key);
    return rec;
});

//Dataset Implementation
Dataset<CsvExtendedRecord> withKeyData = preprocessedData.map((
    MapFunction<CsvRecord,CsvExtendedRecord>) rec -> {
    //Omitted, the same as RDD-based implementation
}, Encoders.bean(CsvExtendedRecord.class));

//DataFrame Implementation
preprocessedData.createOrReplaceTempView("preprocessedData");
Dataset<Row> withKeyData = sparkSQL.sql("SELECT columns CONCAT(
    blocking key definition) as blockingKey FROM preprocessedData");

sparkSQL.udf().register(name, new UDF1<String, String>() {
```

```
    //function implementation omitted
}, DataTypes.StringType);
```

Listing A.4: Blocking

```
//RDD Implementation
JavaPairRDD<String, Record> firstOne = withKeyData
  .mapToPair(new PairFunction<Record, String, Record>() {
    public Tuple2<String, Record> call(Record t) throws Exception {
      return new Tuple2<String, Record>(t.getBlockingKey(), new
    Record(t));
    }
});

JavaRDD<TupleRecord> recordPairData = firstOne.join(firstOne)
  .filter(new Function<Tuple2<String,Tuple2<Record,Record>>,
    Boolean>() {
    //specific implementation omitted
    return result;
  }
}).map(f->f._2)
  .map(new Function<Tuple2<Record,Record>, TupleRecord>() {
    public TupleRecord call(Tuple2<Record, Record> tup) throws
    Exception {
      return new TupleRecord(tup._1, tup._2 );
    }
});

//Dataset Implementation
Dataset<JoinedTuple>  recordPairData =  withKeyData.joinWith(
    withKeyData1,withKeyData.col("blockingKey").equalTo(withKeyData1
    .col("blockingKey")))
  .filter(
    //specific implementation omitted
  ).map((MapFunction<Tuple2<CsvExtendedRecord,CsvExtendedRecord>,
    JoinedTuple>) rec ->{
    JoinedTuple tup = null;
    return new JoinedTuple(rec._1, rec._2);
}, Encoders.bean(JoinedTuple.class));

//DataFrame Implementation
withKeyData.createOrReplaceTempView("blocking");
Dataset<Row>  recordPairData = sparkSQL.sql("SELECT columns,
    renaming columns to name2 FROM blocking r JOIN blocking s on (s.
    blockingKey=r.blockingKey) where r.rec_id > s.rec_id");
```

Listing A.5: Join for generating record pairs

```
//RDD Implementation
JavaRDD<TupleRecord> totalScoreData =  recordPairData.map(new
    Function<TupleRecord, TupleRecord>() {
  public TupleRecord call(TupleRecord tup) throws Exception {
    //calculate scores for each attribute using the selected
    similarity function
    tup.setTotalScore(
      //summing up all scores
    );
    return tup;
```

```
  }
}

JavaRDD<TupleRecord> resultData = totalScoreData.map(new Function<
    TupleRecord, TupleRecord>() {
  public TupleRecord call(TupleRecord rec) throws Exception {
    //classification
    return rec;
  }
});

resultData.saveAsTextFile("path");

//Dataset Implementation
Dataset<CsvExtendedTuple> totalScoreData = recordPairData.map((
    MapFunction<JoinedTuple,CsvExtendedTuple>) rec -> {
  public CsvExtendedTuple call(CsvExtendedTuple tup) throws
    Exception {
    //calculate scores for each attribute using the selected
    similarity function
    tup.setTtotalScore(
      //summing up all scores
    );
    return tup;
}, Encoders.bean(CsvExtendedTuple.class));

Dataset<CsvExtendedTuple> resultData = totalScoreData.map(new
    Function<TupleRecord, TupleRecord>() {
  public TupleRecord call(TupleRecord rec) throws Exception {
    //classification
    return rec;
}, Encoders.bean(TotalScore.class));

resultData.write().csv("path");

//DataFrame Implementation
sparkSQL.udf().register("similarityFunction", new UDF1<String,
    String>() {
  //function implementation omitted
}, DataTypes.StringType);

sparkSQL.udf().register("matchNonmatch", new UDF1<String, String>()
    {
  //function implementation omitted
}, DataTypes.StringType);

Dataset<Row> totalScoreData = recordPairData
  .withColumn("Score", callUDF("similarityFunction", block.col("
    name"), block.col("name2"))
  .withColumn("totalScore", org.apache.spark.sql.functions.expr(
  //summing up all scores
  ).select("rec_id","rec_id2","totalScore");

Dataset<Row> resultData= totalScoreData.withColumn("matchNonmatch",
    callUDF("MatchNonmatch", totalScore.col("totalScore")));

```

```
resultData.write().csv("path");
```

Listing A.6: Similarity calculation and classification

```
//RDD Implementation
JavaRDD<TupleRecord> resultCleanedData =  resultData.map(new
    Function<TupleRecord, TupleRecord>() {
  public TupleRecord call(TupleRecord tup) throws Exception {
    tup.getFirstOrigianl().setCleanI_rec_id(cleanID(tup.
    getFirstOrigianl().getRec_id()));
    tup.getSecondDuplicate().setCleanI_rec_id(cleanID(tup.
    getSecondDuplicate().getRec_id()));
    return tup;
  }
});

long truePositiveCount =  resultCleanedData.filter(tup -> tup.
    getMatchNonmatch().equalsIgnoreCase("match") && tup.
    getFirstOrigianl().getCleanI_rec_id().equalsIgnoreCase(tup.
    getSecondDuplicate().getCleanI_rec_id())).count();
long falsePositiveCount = resultCleanedData
.filter(tup -> tup.getMatchNonmatch().equalsIgnoreCase("match") &&
    !tup.getFirstOrigianl().getCleanI_rec_id().equalsIgnoreCase(tup.
    getSecondDuplicate().getCleanI_rec_id())).count();
long falseNegativeCount = resultCleanedData
.filter(tup -> tup.getMatchNonmatch().equalsIgnoreCase("nonmatch")
    && tup.getFirstOrigianl().getCleanI_rec_id().equalsIgnoreCase(
    tup.getSecondDuplicate().getCleanI_rec_id())).count();

double precision = ((double)truePositiveCount/(truePositiveCount+
    falsePositiveCount);
double recall = ((double)truePositiveCount/(truePositiveCount+
    falseNegativeCount);
double fMeasure = 2*precision*recall/(precision+recall);

String output = precision +"," + recall +"," + fMeasure;

FSDataOutputStream recOutputWriter = null;
FileSystem fs = null;
try {
  Configuration configuration = new Configuration();
  fs = FileSystem.get(new URI(""),configuration);
  recOutputWriter = fs.create(path, true);
  } catch (IOException e) {
  e.printStackTrace();
  }
recOutputWriter.writeChars(output);
if (recOutputWriter != null) {
  recOutputWriter.close();
}
fs.close();
sc.stop();

//Dataset Implementation omitted, similar to RDD-based
    implementation

//DataFrame Implementation
sparkSQL.udf().register("cleanRecID", new UDF1<String, String>(){
```

```
42   public String call(String str) throws Exception {
43     //specific implementation omitted
44   }
45 },DataTypes.StringType);
46
47 Dataset<Row> resultCleanedData= resultData
48   .withColumn("rec_id", callUDF("cleanRecID", totalScore.col("
       rec_id")))
49   .withColumn("rec_id2", callUDF("cleanRecID", totalScore.col("
       rec_id2")));
50
51 resultCleanedData.createOrReplaceTempView("result");
52
53 Dataset<Row> truePositive = sparkSQL.sql("select count(
       matchNonmatch) as truePositive from result where (matchNonmatch
       = 'match' and rec_id = rec_id2)");
54 Dataset<Row> falsePositive = sparkSQL.sql("select count(
       matchNonmatch) as falsePositive from result where (matchNonmatch
        = 'match' and rec_id != rec_id2)");
55 Dataset<Row> falseNegative = sparkSQL.sql("select count(
       matchNonmatch) as falseNegative from result where (matchNonmatch
        = 'nonmatch' and rec_id = rec_id2)");
56
57 Dataset<Row> posNegs = truePositive.crossJoin(falsePositive).
       crossJoin(falseNegative);
58 Dataset<Row> evaluation=  posNegs
59   .withColumn("precision",org.apache.spark.sql.functions.expr("
       truePositive/(truepositive +falsePositive)"))
60   .withColumn("recall", org.apache.spark.sql.functions.expr("
       truePositive/(truepositive + falseNegative)"))
61   .withColumn("F-Measure", org.apache.spark.sql.functions.expr("2*
       precision*recall/(precision+recall)"))
62   .select("precision","recall","F-Measure");
63
64 evaluation.write().csv("path");
65 sparkSQL.stop();
```

Listing A.7: Evaluation

A.3 Imbalance Ratio of Five Load Balancing Strategies

Table A.1 and Table A.2 are the calculated imbalance ratio of the default, BlockSplit, BlockSlicer, Two-Level Split (TLS) and Block-Oriented Slicer (BOS) load balancing strategies with 112 and 224 reducers, respectively. As we can see from both tables, all four tailored load balancing strategies have the same imbalance ratio, which is quite close to the optimal case. The imbalance ratio of the default load balancing strategy is lowered by 112 reducers but is raised by 224 reducers.

Table A.1: Imbalance ratio of different strategies with 112 reducers.

	$10^6 + 50\%$	$2 * 10^6 + 05\%$	DBLP_0.5	DBLP_1
Default	6.409	6.423	19.376	18.582
BlockSplit	1.000	1.000	1.000	1.000
BlockSlicer	1.000	1.000	1.000	1.000
TLS	1.000	1.000	1.000	1.000
BOS	1.000	1.000	1.000	1.000

Table A.2: Imbalance ratio of different strategies with 224 reducers.

	$10^6 + 50\%$	$2 * 10^6 + 05\%$	DBLP_0.5	DBLP_1
Default	12.557	12.589	38.686	37.092
BlockSplit	1.000	1.000	1.000	1.000
BlockSlicer	1.000	1.000	1.000	1.000
TLS	1.000	1.000	1.000	1.000
BOS	1.000	1.000	1.000	1.000

Bibliography

Azza Abouzeid, Kamil Bajda-Pawlikowski, Daniel Abadi, Avi Silberschatz, and Alexander Rasin. HadoopDB: An Architectural Hybrid of Mapreduce and Dbms Technologies for Analytical Workloads. *Proceedings of the VLDB Endowment*, 2 (1):922–933, 2009. (cited on Page 22 and 49)

Akiko Aizawa and Keizo Oyama. A Fast Linkage Detection Scheme for Multi-source Information Integration. In *Proceedings of the International Workshop on Challenges in Web Information Retrieval and Integration (WIRI)*, pages 30–39. IEEE, 2005. (cited on Page 8)

Amer Al-Badarneh. Join Algorithms under Apache Spark: Revisited. In *Proceedings of the International Conference on Computer and Technology Applications (ICCTA)*, pages 56–62, 2019. (cited on Page 46)

Arvind Arasu, Michaela Götz, and Raghav Kaushik. On Active Learning of Record Matching Packages. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 783–794, 2010. (cited on Page 2 and 111)

Michael Armbrust, Reynold S Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K Bradley, Xiangrui Meng, Tomer Kaftan, Michael J Franklin, Ali Ghodsi, et al. Spark Sql: Relational Data Processing in Spark. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 1383–1394. ACM, 2015. (cited on Page xiii and 17)

Fariha Atta, Stratis D Viglas, and Salman Niazi. Sand Join—a Skew Handling Join Algorithm for Google's Mapreduce Framework. In *Proceedings of the International Multitopic conference (INMIC)*, pages 170–175. IEEE, 2011. (cited on Page 83)

Ranieri Baraglia, Gianmarco De Francisci Morales, and Claudio Lucchese. Document Similarity Self-join with Mapreduce. In *Proceedings of the International Conference on Data Mining (ICDM)*, pages 731–736. IEEE, 2010. (cited on Page 20, 25, 26, 29, 35, and 42)

C Bell, Ian H Witten, and A Moffat. Managing Gigabytes: Compressing and Indexing Documents and Images. *IEEE Transactions on Information Theory*, 1999. (cited on Page 38)

Kedar Bellare, Suresh Iyengar, Aditya G. Parameswaran, and Vibhor Rastogi. Active Sampling for Entity Matching. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1131–1139, 2012. (cited on Page 111)

Kedar Bellare, Suresh Iyengar, Aditya G. Parameswaran, and Vibhor Rastogi. Active Sampling for Entity Matching with Guarantees. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, pages 12:1–12:24, 2013. (cited on Page 2 and 111)

Omar Benjelloun, Hector Garcia-Molina, Heng Gong, Hideki Kawai, Tait E Larson, David Menestrina, and Sutthipong Thavisomboon. D-swoosh: A Family of Algorithms for Generic, Distributed Entity Resolution. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, pages 37–37. IEEE, 2007. (cited on Page 20, 23, 28, 33, 41, and 49)

S Prabhakar Benny, S Vasavi, and P Anupriya. Hadoop Framework for Entity Resolution within High Velocity Streams. *Procedia Computer Science*, 85:550–557, 2016. (cited on Page 21, 24, 31, 37, and 42)

Christopher M Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006. (cited on Page 11)

Christoph Böhm, Gerard de Melo, Felix Naumann, and Gerhard Weikum. Linda: Distributed Web-of-data-scale Entity Matching. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, pages 2104–2108. ACM, 2012. (cited on Page 20, 23, 24, 25, 28, 29, 32, and 36)

P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. Enriching Word Vectors with Subword Information. *arXiv preprint arXiv:1607.04606*, 2016. (cited on Page 10 and 90)

Christine L Borgman and Susan L Siegfried. Getty's Synoname™ and Its Cousins: A Survey of Applications of Personal Name-matching Algorithms. *Journal of the American Society for Information Science*, 43(7):459–476, 1992. (cited on Page 9)

Ron Bowes. Facebook Names Dataset. Available at: https://blog.skullsecurity.org/2010/return-of-the-facebook-snatchers, 2010. (cited on Page 59)

David Guy Brizan and Abdullah Uz Tansel. A Survey of Entity Resolution and Record Linkage Methodologies. *Communications of the IIMA (CIIMA)*, 6(3):5, 2006. (cited on Page 6, 45, and 95)

Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache Flink: Stream and Batch Processing in a Single Engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering (IEEE-CS)*, 36(4), 2015. (cited on Page 43)

Dehua Chen, Changgan Shen, Jieying Feng, and Jiajin Le. An Efficient Parallel Top-k Similarity Join for Massive Multidimensional Data Using Spark. *International Journal of Database Theory and Application*, 8(3):57–68, 2015. (cited on Page 21, 22, 25, 27, 32, 38, and 62)

Gang Chen, Keyu Yang, Lu Chen, Yunjun Gao, Baihua Zheng, and Chun Chen. Metric Similarity Joins Using Mapreduce. *IEEE Transactions on Knowledge and*

Data Engineering (TKDE), 29(3):656–669, 2017. (cited on Page 21, 26, 31, 37, and 42)

Qi Chen, Jinyu Yao, and Zhen Xiao. Libra: Lightweight Data Skew Mitigation in Mapreduce. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 26 (9):2520–2533, 2014. (cited on Page 83)

Tianqi Chen and Carlos Guestrinand. Xgboost: A Scalable Tree Boosting System. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 785–794. ACM, 2016. (cited on Page 12)

Xiao Chen, Kirity Rapuru, Gabriel Durand, Eike Schallehn, and Gunter Saake. Performance Comparison of Three Spark-based Implementations of Parallel Entity Resolution. In *Proceedings of the International Workshop on Big Data Management in Cloud Systems (DEXA-BDMICS)*, pages 76–87. Springer, 2018a. (cited on Page 5 and 49)

Xiao Chen, Eike Schallehn, and Gunter Saake. Cloud-scale Entity Resolution: Current State and Open Challenges. *Open Journal of Big Data (OJBD)*, 4(1):30–51, 2018b. (cited on Page 1, 5, 19, 65, 66, and 99)

Xiao Chen, Roman Zoun, Eike Schallehn, Saravani Mantha, Kirity Rapuru, and Gunter Saake. Exploring Spark-sql-based Entity Resolution Using the Persistence Capability. In *Proceedings of the International Conference: Beyond Databases, Architectures and Structures (BDAS)*, pages 3–17. Springer, 2018c. (cited on Page 49)

Xiao Chen, Gabriel Campero Durand, Roman Zoun, David Broneske, Yang Li, and Gunter Saake. The Best of Both Worlds: Combining Hand-tuned and Word-embedding-based Similarity Measures for Entity Resolution. In *In Fachtagung Datenbanksysteme für Business, Technologie und Web (BTW)*, 2019a. (cited on Page 87 and 99)

Xiao Chen, Yinlong Xu, David Broneske, Gabriel Campero Durand, Roman Zoun, and Gunter Saake. Heterogeneous Committee-based Active Learning for Entity Resolution (HeALER). In *Proceedings of the European Conference on Advances in Databases and Information Systems (ADBIS)*, pages 69–85. Springer, 2019b. (cited on Page 97)

Xiao Chen, Nishanth Entoor Venkatarathnam, Rapuru Kirity, David Broneske, Gabriel Campero Durand, Roman Zoun, and Gunter Saake. Analysis and comparison of block-splitting-based load balancing strategies for parallel entity resolution. In *Proceedings of the International Conference on Information Integration and Web-based Applications & Services(iiWAS)*. ACM, 2020. (cited on Page 65)

Peter Christen. A Comparison of Personal Name Matching: Techniques and Practical Issues. In *Proceedings of the International Conference on Data Mining Workshops (ICDMW)*, pages 290–294. IEEE, 2006. (cited on Page 9)

Peter Christen. Febrl – an Open Source Data Cleaning, Deduplication and Record Linkage System with a Graphical User Interface. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1065–1068. ACM, 2008. (cited on Page 20, 23, 27, 28, and 33)

Peter Christen. A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 24(9):1537–1555, 2012a. (cited on Page 1, 6, 7, 9, 38, and 45)

Peter Christen. *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Springer Science & Business Media, 2012b. (cited on Page xiii, 1, 5, 6, 7, 9, and 10)

Peter Christen and Dinusha Vatsalan. Flexible and Extensible Generation and Corruption of Personal Data. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, pages 1165–1168. ACM, 2013. (cited on Page 18)

Vassilis Christophides, Vasilis Efthymiou, Themis Palpanas, George Papadakis, and Kostas Stefanidis. End-to-end Entity Resolution for Big Data: A Survey. *arXiv preprint arXiv:1905.06397*, 2019. (cited on Page 46)

Xu Chu, Ihab F Ilyas, and Paraschos Koutris. Distributed Data Deduplication. *Proceedings of the VLDB Endowment*, 9(11):864–875, 2016. (cited on Page 21, 24, 31, and 37)

Tim Churches, Peter Christen, Kim Lim, and Justin Xi Zhu. Preparation of Name and Address Data for Record Linkage Using Hidden Markov Models. *BMC Medical Informatics and Decision Making*, 2(1):9, 2002. (cited on Page 6)

William W Cohen, Pradeep Ravikumar, Stephen E Fienberg, et al. A Comparison of String Distance Metrics for Name-matching Tasks. In *Proceedings of the International Workshop on Information Integration on the Web (IIWeb)*, volume 2003, pages 73–78, 2003. (cited on Page 9, 52, 95, and 96)

Thomas Cover and Peter Hart. Nearest Neighbor Pattern Classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967. (cited on Page 11)

Guilherme Dal Bianco, Renata Galante, and Carlos A Heuser. A Fast Approach for Parallel Deduplication on Multicore Processors. In *Proceedings of the ACM Symposium on Applied Computing (SAC)*, pages 1027–1032. ACM, 2011. (cited on Page 20, 23, 24, 29, 35, and 39)

Fred J Damerau. A Technique for Computer Detection and Correction of Spelling Errors. *Communications of the ACM*, 7(3):171–176, 1964. (cited on Page 9)

Akash Das Sarma, Yeye He, and Surajit Chaudhuri. Clusterjoin: A Similarity Joins Framework Using Map-Reduce. *Proceedings of the VLDB Endowment*, 7(12): 1059–1070, 2014. (cited on Page 21, 26, 30, 36, 41, and 42)

Junio de Freitas, Gisele L. Pappa, Altigran Soares da Silva, et al. Active Learning Genetic Programming for Record Deduplication. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, 2010. (cited on Page 2, 107, and 111)

Dong Deng, Guoliang Li, Shuang Hao, Jiannan Wang, and Jianhua Feng. Massjoin: A Mapreduce-based Method for Scalable String Similarity Joins. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 340–351. IEEE, 2014. (cited on Page 9, 21, 25, 26, 28, 30, 36, 41, and 42)

David J DeWitt, Jeffrey F Naughton, Donovan A Schneider, and Srinivasan Seshadri. Practical Skew Handling in Parallel Joins. Technical report, University of Wisconsin-Madison, Department of Computer Sciences, 1992. (cited on Page 83)

Lee R Dice. Measures of the Amount of Ecologic Association between Species. *Ecology*, 26(3):297–302, 1945. (cited on Page 9)

Martin Dillon. Introduction to modern information retrieval: G. salton and m. mcgill. mcgraw-hill, new york (1983). xv + 448 pp., $32.95 isbn 0-07-054484-0. *Information Processing and Management*, 19:402–403, 1983. (cited on Page 9)

Xin Dong, Alon Halevy, and Jayant Madhavan. Reference Reconciliation in Complex Information Spaces. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 85–96, 2005. (cited on Page 5)

Finale Doshi-Velez and Been Kim. Towards a Rigorous Science of Interpretable Machine Learning. *arXiv preprint arXiv:1702.08608*, 2017. (cited on Page 103)

Chenxiao Dou, Yi Cui, Daniel Sun, Raymond Wong, Muhammad Atif, Guoqiang Li, and Rajiv Ranjan. Unsupervised Blocking and Probabilistic Parallelisation for Record Matching of Distributed Big Data. *The Journal of Supercomputing*, 75(2): 623–645, 2019. (cited on Page 21, 22, 25, 27, 28, 32, 38, and 62)

Uwe Draisbach, Felix Naumann, Sascha Szott, and Oliver Wonneberg. Adaptive Windows for Duplicate Detection. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 1073–1083. IEEE, 2012. (cited on Page 11, 36, and 38)

Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafi Joty, Mourad Ouzzani, and Nan Tang. Distributed Representations of Tuples for Entity Resolution. *Proceedings of the VLDB Endowment*, pages 1454–1467, 2018. (cited on Page 2, 10, 87, 95, and 116)

Vasilis Efthymiou, George Papadakis, George Papastefanatos, Kostas Stefanidis, and Themis Palpanas. Parallel Meta-blocking: Realizing Scalable Entity Resolution Over Large, Heterogeneous Data. In *Proceedings of the International Conference on Big Data (Big Data)*, pages 411–420. IEEE, 2015. (cited on Page 21)

Vasilis Efthymiou, George Papadakis, George Papastefanatos, Kostas Stefanidis, and Themis Palpanas. Parallel Meta-blocking for Scaling Entity Resolution Over Big Heterogeneous Data. *Information Systems*, 65:137–157, 2017. (cited on Page 1, 21, 24, 31, 37, and 42)

Randa M Abd El-Ghafar, Mervat H Gheith, Ali H El-Bastawissy, and Eman S Nasr. Record Linkage Approaches in Big Data: A State of Art Study. In *International Computer Engineering Conference (ICENCO)*, pages 224–230. IEEE, 2017. (cited on Page 46)

Mohamed G Elfeky, Vassilios S Verykios, and Ahmed K Elmagarmid. Tailor: A Record Linkage Toolbox. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 17–28. IEEE, 2002. (cited on Page 8)

Ahmed K Elmagarmid, Panagiotis G Ipeirotis, and Vassilios S Verykios. Duplicate Record Detection: A Survey. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 19(1):1–16, 2007. (cited on Page 1, 45, and 100)

Tamer Elsayed, Jimmy Lin, and Douglas W Oard. Pairwise Document Similarity in Large Collections with Mapreduce. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics on Human Language Technologies (HLT)*, pages 265–268, 2008. (cited on Page 20, 25, 26, 29, and 35)

Ivan P Fellegi and Alan B Sunter. A Theory for Record Linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969. (cited on Page 1, 5, and 7)

Fabian Fier, Nikolaus Augsten, Panagiotis Bouros, Ulf Leser, and Johann-Christoph Freytag. Set Similarity Joins on Mapreduce: An Experimental Survey. *Proceedings of the VLDB Endowment*, 11(10):1110–1122, 2018. (cited on Page 46)

Jeffrey Fisher, Peter Christen, and Qing Wang. Active Learning Based Entity Resolution Using Markov Logic. In *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, pages 338–349, 2016. (cited on Page 111)

Ellen Friedman and Kostas Tzoumas. *Introduction to Apache Flink: Stream Processing for Real Time and Beyond*. O'Reilly Media, Inc., 2016. (cited on Page 43)

Sergej Fries, Brigitte Boden, Grzegorz Stepien, and Thomas Seidl. Phidj: Parallel Similarity Self-join for High-dimensional Vector Data with Mapreduce. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 796–807. IEEE, 2014. (cited on Page 20, 25, 26, 30, 36, 41, and 42)

Avigdor Gal. Uncertain Entity Resolution: Re-evaluating Entity Resolution in the Big Data Era: Tutorial. *Proceedings of the VLDB Endowment*, 7(13):1711–1712, 2014. (cited on Page 46)

Elaheh Gavagsaz, Ali Rezaee, and Hamid Haj Seyyed Javadi. Load Balancing in Reducers for Skewed Data in Mapreduce Systems by Using Scalable Simple Random Sampling. *The Journal of Supercomputing*, 74:3415–3440, 2018. (cited on Page 83)

Lise Getoor and Ashwin Machanavajjhala. Entity Resolution: Theory, Practice & Open Challenges. *Proceedings of the VLDB Endowment*, 5(12):2018–2019, 2012. (cited on Page 45)

Lise Getoor and Ashwin Machanavajjhala. Entity Resolution for Big Data. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1527–1527. ACM, 2013. (cited on Page 13 and 45)

Leicester E Gill. Ox-link: The Oxford Medical Record Linkage System. In *Proceedings of the International Workshop and Exposition: Record Linkage Techniques*, pages 15–33. Citeseer, 1997. (cited on Page 9)

Demetrio Gomes Mestre and Carlos Eduardo Santos Pires. Improving Load Balancing for Mapreduce-based Entity Matching. In *Proceedings of the Symposium on Computers and Communications (ISCC)*, pages 000618–000624. IEEE, 2013. (cited on Page 2, 21, 24, 30, 36, 40, 41, 42, 68, 69, and 84)

Lifang Gu, Rohan Baxter, Deanne Vickers, and Chris Rainsford. Record Linkage: Current Practice and Future Directions. Technical report, Department of Mathematical and Information Sciences, CSIRO, 2003. (cited on Page 44)

Benjamin Gufler, Nikolaus Augsten, Angelika Reiser, and Alfons Kemper. Handling Data Skew in Mapreduce. In *Proceedings of the International Conference on Cloud Computing and Services Science (CLOSER)*, volume 11, pages 574–583, 2011. (cited on Page 83)

Benjamin Gufler, Nikolaus Augsten, Angelika Reiser, and Alfons Kemper. Load Balancing in Mapreduce Based on Scalable Cardinality Estimates. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 522–533. IEEE, 2012. (cited on Page 83)

David Hand and Peter Christen. A Note on Using the F-measure for Evaluating Record Linkage Algorithms. *Statistics and Computing*, 28(3):539–547, 2018. (cited on Page 14)

M Al Hajj Hassan, Mostafa Bamha, and Frédéric Loulergue. Handling Data-skew Effects in Join Operations Using Mapreduce. *Procedia Computer Science*, 29: 145–158, 2014. (cited on Page 83)

Mauricio A Hernández and Salvatore J Stolfo. The Merge/purge Problem for Large Databases. In *ACM Sigmod Record*, pages 127–138. ACM, 1995. (cited on Page 6, 7, 11, and 38)

Mauricio A Hernández and Salvatore J Stolfo. Real-world Data Is Dirty: Data Cleansing and the Merge/purge Problem. *Data Mining and Knowledge Discovery*, 2(1):9–37, 1998. (cited on Page 6)

Melanie Herschel, Felix Naumann, Sascha Szott, and Maik Taubert. Scalable Iterative Graph Duplicate Detection. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 24(11):2094–2108, 2012. (cited on Page 20, 23, 25, 28, 32, 33, and 44)

David Holmes and M Catherine McCabe. Improving Precision and Recall for Soundex Retrieval. In *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC)*, pages 22–26. IEEE, 2002. (cited on Page 9)

Hortonworks. Hortonworks Data Platform. Available at: https://www.cloudera.com/products/hdp.html, 2020. (cited on Page 59)

Sue-Chen Hsueh, Ming-Yen Lin, and Yi-Chun Chiu. A Load-balanced Mapreduce Algorithm for Blocking-based Entity-resolution with Multiple Keys. In *Proceedings of the Australasian Symposium on Parallel and Distributed Computing (AusPDC)*, pages 3–9, 2014. (cited on Page 2, 20, 41, and 84)

Timothy Huang and Stuart Russell. Object Identification: A Bayesian Analysis with Application to Traffic Surveillance. *Artificial Intelligence*, 103(1-2):77–93, 1998. (cited on Page 5)

Shadi Ibrahim, Hai Jin, Lu Lu, Song Wu, Bingsheng He, and Li Qi. Leen: Locality/fairness-aware Key Partitioning for Mapreduce in the Cloud. In *Proceedings of the International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 17–24. IEEE, 2010. (cited on Page 83)

Shadi Ibrahim, Hai Jin, Lu Lu, Bingsheng He, Gabriel Antoniu, and Song Wu. Handling Partitioning Skew in Mapreduce Using Leen. *Peer-to-Peer Networking and Applications*, 6(4):409–424, 2013. (cited on Page 83)

Bhattacharya Indrajit and Getoor Lise. Collective Entity Resolution in Relational Data. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):5, 2006. (cited on Page 6)

Piotr Indyk and Rajeev Motwani. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 604–613. ACM, 1998. (cited on Page 38)

Robert Isele and Christian Bizer. Active Learning of Expressive Linkage Rules Using Genetic Programming. *Journal of Web Semantics*, pages 2–15, 2013. (cited on Page 2 and 111)

Edwin H Jacox and Hanan Samet. Metric Space Similarity Joins. *ACM Transactions on Database Systems (TODS)*, 33(2):7, 2008. (cited on Page 35 and 39)

Miyoung Jang and Jae-Woo Chang. Grid-based Parallel Algorithms of Join Queries for Analyzing Multi-dimensional Data on Mapreduce. *IEICE Transactions on Information and Systems*, 101(4):964–976, 2018. (cited on Page 21, 25, 26, 31, 32, 37, and 41)

Matthew A Jaro. *Unimatch, a Record Linkage System: Users Manual*. Bureau of the Census, 1980. (cited on Page 9)

Yu Jiang, Dong Deng, Jiannan Wang, Guoliang Li, and Jianhua Feng. Efficient Parallel Partition-based Algorithms for Similarity Search and Join with Edit Distance Constraints. In *EDBT/ICDT Workshops*, pages 341–348. ACM, 2013. (cited on Page 20, 23, 25, 28, 33, and 42)

Yu Jiang, Guoliang Li, Jianhua Feng, and Wen-Syan Li. String Similarity Joins: An Experimental Evaluation. *Proceedings of the VLDB Endowment*, 7(8):625–636, 2014. (cited on Page 45)

Liang Jin, Chen Li, and Sharad Mehrotra. Efficient Record Linkage in Large Data Sets. In *Proceedings of the International Conference on Database Systems for Advanced Applications (DASFAA)*, pages 137–146. IEEE, 2003. (cited on Page 8)

Dimitrios Karapiperis and Vassilios S Verykios. Load-balancing the Distance Computations in Record Linkage. *ACM SIGKDD Explorations Newsletter*, 17(1):1–7, 2015. (cited on Page 2 and 84)

Holden Karau and Rachel Warren. *High Performance Spark*. O'Reilly Media, 2017. (cited on Page 17)

Hakan Kardes, Deepak Konidena, Siddharth Agrawal, Micah Huff, and Ang Sun. Graph-based Approaches for Organization Entity Resolution in Mapreduce. In *Proceedings of the Workshop on Graph-Based Methods for Natural Language Processing (TextGraphs)*, page 70, 2013. (cited on Page 20, 24, 28, 30, 32, 36, and 44)

Hideki Kawai, Hector Garcia-Molina, Omar Benjelloun, David Menestrina, Euijong Whang, and Heng Gong. P-Swoosh: Parallel Algorithm for Generic Entity Resolution. Technical report, Stanford InfoLab, 2006. (cited on Page 20, 23, 28, 33, and 49)

Hung-sik Kim and Dongwon Lee. Parallel Linkage. In *Proceedings of the International Conference on Information and Knowledge Management (cikm)*, pages 283–292. ACM, 2007. (cited on Page 20, 23, 25, 28, and 33)

Younghoon Kim and Kyuseok Shim. Parallel Top-k Similarity Join Algorithms Using Mapreduce. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 510–521. IEEE, 2012. (cited on Page 20, 26, 29, 32, and 35)

Toralf Kirsten, Lars Kolb, Michael Hartung, Anika Groß, Hanna Köpcke, and Erhard Rahm. Data Partitioning for Parallel Entity Matching. *Proceedings of the VLDB Endowment*, 2010. (cited on Page 20 and 39)

Lars Kolb and Erhard Rahm. Parallel Entity Resolution with Dedoop. *Datenbank-Spektrum*, 13(1):23–32, 2013. (cited on Page 20)

Lars Kolb, Hanna Köpcke, Andreas Thor, and Erhard Rahm. Learning-based Entity Resolution with Mapreduce. In *Proceedings of the International Workshop on Cloud Data Management (CloudDB)*, pages 1–6. ACM, 2011a. (cited on Page 5 and 20)

Lars Kolb, Andreas Thor, and Erhard Rahm. Block-based Load Balancing for Entity Resolution with Mapreduce. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, pages 2397–2400. ACM, 2011b. (cited on Page 20)

Lars Kolb, Andreas Thor, and Erhard Rahm. Dedoop: Efficient Deduplication with Hadoop. *Proceedings of the VLDB Endowment*, 5(12):1878–1881, 2012a. (cited on Page 2, 20, 24, 29, 34, 35, and 41)

Lars Kolb, Andreas Thor, and Erhard Rahm. Load Balancing for Mapreduce-based Entity Resolution. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 618–629. IEEE, 2012b. (cited on Page 2, 20, 40, 41, 66, 68, 69, and 84)

Lars Kolb, Andreas Thor, and Erhard Rahm. Multi-pass Sorted Neighborhood Blocking with Mapreduce. *Computer Science-Research and Development (CSRD)*, 27(1):45–63, 2012c. (cited on Page 20 and 41)

Lars Kolb, Andreas Thor, and Erhard Rahm. Don't Match Twice: Redundancy-free Similarity Computation with Mapreduce. In *Proceedings of the Workshop on Data Analytics in the Cloud (DanaC)*, pages 1–5. ACM, 2013. (cited on Page 20)

N. Kooli, R. Allesiardo, and E. Pigneul. Deep Learning Based Approach for Entity Resolution in Databases. In *Proceedings of the Asian Conference on Intelligent Information and Database Systems (ACIIDS)*, pages 3–12. Springer, 2018. (cited on Page 2, 10, 87, and 95)

H. Köpcke and E. Rahm. Training Selection for Tuning Entity Matching. In *Proceedings of the International Workshop on Quality in Databases and Management of Uncertain Data (QDB/MUD)*, pages 3–12, 2008. (cited on Page 116)

Hanna Köpcke and Erhard Rahm. Frameworks for Entity Matching: A Comparison. *Data & Knowledge Engineering (DKE)*, 69(2):197–210, 2010. (cited on Page 45)

Hanna Köpcke, Andreas Thor, and Erhard Rahm. Evaluation of Entity Resolution Approaches on Real-world Match Problems. *Proceedings of the VLDB Endowment*, 3(1-2):484–493, 2010. (cited on Page 96)

Sotiris Kotsiantis, Dimitris Kanellopoulos, Panayiotis Pintelas, et al. Handling Imbalanced Datasets: A Review. *GESTS International Transaction on Computer Science and Engineering*, 30(1):25–36, 2006. (cited on Page 98)

Nick Koudas, Sunita Sarawagi, and Divesh Srivastava. Record Linkage: Similarity Measures and Algorithms. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 802–803, 2006. (cited on Page 45 and 95)

YongChul Kwon, Magdalena Balazinska, Bill Howe, and Jerome Rolia. A Study of Skew in Mapreduce Applications. *Open Cirrus Summit*, 11, 2011. (cited on Page 82)

YongChul Kwon, Magdalena Balazinska, Bill Howe, and Jerome Rolia. Skewtune: Mitigating Skew in Mapreduce Applications. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 25–36. ACM, 2012. (cited on Page 83)

Andrew J Lait and Brian Randell. An Assessment of Name Matching Algorithms. *Technical Report, Department of Computing Science, Series-University of Newcastle Upon Tyne*, 1996. (cited on Page 9)

Vladimir I Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions, and Reversals. In *Soviet Physics Doklady*, volume 10, pages 707–710, 1966. (cited on Page 9)

Jianjiang Li, Yajun Liu, Jian Pan, Peng Zhang, Wei Chen, and Lizhe Wang. Map-balance-reduce: An Improved Parallel Programming Model for Load Balancing of Mapreduce. *Future Generation Computer Systems (FGCS)*, 2017. (cited on Page 83)

Wen Liu, Yanming Shen, and Peng Wang. An Efficient Mapreduce Algorithm for Similarity Join in Metric Spaces. *The Journal of Supercomputing*, 72(3):1179–1200, 2016. (cited on Page 21, 26, 30, 37, and 42)

Wei Lu, Yanyan Shen, Su Chen, and Beng Chin Ooi. Efficient Processing of K Nearest Neighbor Joins Using Mapreduce. *Proceedings of the VLDB Endowment*, 5(10):1016–1027, 2012. (cited on Page 20, 26, 30, 32, 36, and 42)

Z. Lu, X. Wu, and J. Bongard. Active Learning with Adaptive Heterogeneous Ensembles. In *Proceedings of the International Conference on Data Mining (ICDM)*, pages 327–336, 2009. (cited on Page 97)

Wuman Luo, Haoyu Tan, Huajian Mao, and Lionel M Ni. Efficient Similarity Joins on Massive High-dimensional Datasets Using Mapreduce. In *Proceedings of the International Conference on Mobile Data Management (MDM)*, pages 1–10. IEEE, 2012. (cited on Page 20, 25, 26, 29, and 35)

Kun Ma and Bo Yang. Parallel Nosql Entity Resolution Approach with Mapreduce. In *Proceedings of the International Conference on Intelligent Networking and Collaborative Systems (INCOS)*, pages 384–389. IEEE, 2015. (cited on Page 21 and 24)

Kun Ma, Fusen Dong, and Bo Yang. Large-scale Schema-free Data Deduplication Approach with Adaptive Sliding Window Using Mapreduce. *The Computer Journal*, 58(11):3187–3201, 2015. (cited on Page 21, 30, and 37)

Youzhong Ma, Xiaofeng Meng, and Shaoya Wang. Parallel Similarity Joins on Massive High-dimensional Data Using Mapreduce. *Concurrency and Computation: Practice and Experience*, 28(1):166–183, 2016. (cited on Page 21, 25, 26, 30, 37, and 42)

John Makhoul, Francis Kubala, Richard Schwartz, Ralph Weischedel, et al. Performance Measures for Information Extraction. In *Proceedings of Broadcast News Workshop (DARPA)*, pages 249–252, 1999. (cited on Page 14)

Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: A System for Large-scale Graph Processing. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 135–146. ACM, 2010. (cited on Page 44)

Pankaj Malhotra, Puneet Agarwal, and Gautam Shroff. Graph-parallel Entity Resolution Using Lsh & Imm. In *EDBT/ICDT Workshops*, pages 41–49, 2014. (cited on Page 21, 23, 24, 30, 32, 36, 42, and 44)

Naoki Abe Hiroshi Mamitsuka et al. Query Learning Strategies Using Boosting and Bagging. In *Proceedings of the International Conference on Machine Learning (ICML)*, 1998. (cited on Page 98)

Willi Mann, Nikolaus Augsten, and Panagiotis Bouros. An Empirical Evaluation of Set Similarity Join Techniques. *Proceedings of the VLDB Endowment*, 9(9): 636–647, 2016. (cited on Page 45)

Andrew McCallum, Kamal Nigam, and Lyle H Ungar. Efficient Clustering of High-dimensional Data Sets with Application to Reference Matching. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 169–178. ACM, 2000. (cited on Page 8 and 43)

N McNeill, Hakan Kardes, and Andrew Borthwick. Dynamic Record Blocking: Efficient Linking of Massive Databases in Mapreduce. In *Proceedings of the International Workshop on Quality in Databases (QDB)*, 2012. (cited on Page 2, 39, and 84)

Prem Melville and Raymond J. Mooney. Diverse Ensembles for Active Learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2004. (cited on Page 97)

Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, Db Tsai, Manish Amde, and Sean Owen. MLlib: Machine Learning in Apache Spark. *Journal of Machine Learning Research*, 17(1):1235–1241, 2015. (cited on Page 17 and 103)

Demetrio Gomes Mestre, Carlos Eduardo Pires, and Dimas C. Nascimento. Adaptive Sorted Neighborhood Blocking for Entity Matching with Mapreduce. In *Proceedings of the Symposium on Applied Computing (SAC)*, pages 981–987, New York, NY, USA, 2015. ACM. (cited on Page 21 and 42)

Demetrio Gomes Mestre, Carlos Eduardo Santos Pires, Dimas Cassimiro Nascimento, Andreza Raquel Monteiro de Queiroz, Veruska Borges Santos, and Tiago Brasileiro Araujo. An Efficient Spark-based Adaptive Windowing for Entity Matching. *Journal of Systems and Software*, 128:1–10, 2017. (cited on Page 21, 27, 32, 38, 42, and 62)

Ahmed Metwally and Christos Faloutsos. V-smart-join: A Scalable Mapreduce Framework for All-pair Similarity Joins of Multisets and Vectors. *Proceedings of the VLDB Endowment*, 5(8):704–715, 2012. (cited on Page 20, 26, 29, and 35)

T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient Estimation of Word Representations in Vector Space. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2013a. (cited on Page 10)

T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed Representations of Words and Phrases and Their Compositionality. In *Proceedings of the International Conference on Neural Information Processing Systems (NIPS)*. Curran Associates, 2013b. (cited on Page 90)

Thomas M Mitchell et al. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1997. (cited on Page 11)

Alvaro E Monge, Charles Elkan, et al. The Field Matching Problem: Algorithms and Applications. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*, volume 2, pages 267–270, 1996. (cited on Page 9)

S. Mudgal, H. Li, T. Rekatsinas, A. Doan, Y. Park, G. Krishnan, R. Deep, E. Arcaute, and V. Raghavendra. Deep Learning for Entity Matching: A Design Space Exploration. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 19–34. ACM, 2018. (cited on Page 2, 10, 87, 95, and 96)

Alexandros Nanopoulos, Yannis Manolopoulos, and Yannis Theodoridis. An Efficient and Effective Algorithm for Density Biased Sampling. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, pages 398–404. ACM, 2002. (cited on Page 117)

Gonzalo Navarro, Erkki Sutinen, Jani Tanninen, and Jorma Tarhio. Indexing Text with Approximate Q-grams. In *Proceedings of the Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 350–363. Springer, 2000. (cited on Page 8, 9, and 38)

Howard B Newcombe and James M Kennedy. Record Linkage: Making Maximum Use of the Discriminating Power of Identifying Information. *Communications of the ACM*, 5(11):563–566, 1962. (cited on Page 10 and 95)

Howard B Newcombe, James M Kennedy, SJ Axford, and Allison P James. Automatic Linkage of Vital Records. *Science*, 130(3381):954–959, 1959. (cited on Page 10)

Axel-Cyrille Ngonga Ngomo and Klaus Lyko. EAGLE: Efficient Active Learning of Link Specifications Using Genetic Programming. In *Proceedings of the European Semantic Web Conference (ESWC)*, pages 149–163, 2012. (cited on Page 111)

Axel-Cyrille Ngonga Ngomo, Jens Lehmann, Sören Auer, and Konrad Höffner. RAVEN - Active Learning of Link Specifications. In *Proceedings of the International Workshop on Ontology Matching (OM)*, 2011. (cited on Page 111)

Axel-Cyrille Ngonga Ngomo, Klaus Lyko, and Victor Christen. COALA - Correlation-aware Active Learning of Link Specifications. In *Proceedings of the European Semantic Web Conference (ESWC)*, pages 442–456, 2013. (cited on Page 111)

Hieu T Nguyen and Arnold Smeulders. Active Learning Using Pre-clustering. In *Proceedings of the International Conference on Machine Learning (ICML)*, page 79, 2004. (cited on Page 103)

M Odell and R Russell. The Soundex Coding System. *US Patents*, 1261167, 1918. (cited on Page 9)

Alper Okcan and Mirek Riedewald. Processing Theta-joins Using Mapreduce. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 949–960. ACM, 2011. (cited on Page 83)

J Scott Olsson and Douglas W Oard. Improving Text Classification for Oral History Archives with Temporal Domain Knowledge. In *Proceedings of the International Conference on Research and Development in Information Retrieval (SIGIR)*, pages 623–630. ACM, 2007. (cited on Page 29)

George Papadakis and Themis Palpanas. Blocking for Large-scale Entity Resolution: Challenges, Algorithms, and Practical Examples. In *International Conference on Data Engineering (ICDE)*, pages 1436–1439. IEEE, 2016. (cited on Page 46)

George Papadakis, Jonathan Svirsky, Avigdor Gal, and Themis Palpanas. Comparative Analysis of Approximate Blocking Techniques for Entity Resolution. *Proceedings of the VLDB Endowment*, 9(9):684–695, 2016. (cited on Page 1, 38, and 45)

George Papadakis, Konstantina Bereta, Themis Palpanas, and Manolis Koubarakis. Multi-core Meta-blocking for Big Linked Data. In *Proceedings of the International Conference on Semantic Systems (I-SEMANTICS)*, pages 33–40, 2017. (cited on Page 21)

Andrew Pavlo, Erik Paulson, Alexander Rasin, Daniel J Abadi, David J DeWitt, Samuel Madden, and Michael Stonebraker. A Comparison of Approaches to Large-scale Data Analysis. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 165–178. ACM, 2009. (cited on Page 2, 22, and 34)

J. Pennington and Christopher R. Socher, Riand Manning. Glove: Global Vectors for Word Representation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. (cited on Page 10 and 95)

Lawrence Philips. The Double Metaphone Search Algorithm. *C/C++ Users Journal*, 18(6):38–43, 2000. (cited on Page 9)

Robespierre Pita, Clicia Pinto, Pedro Melo, Malu Silva, Marcos Barreto, and Davide Rasella. A Spark-based Workflow for Probabilistic Record Linkage of Healthcare Data. In *Proceedings of the Workshop on Algorithms and Systems for MapReduce and Beyond (BeyondMR)*, pages 17–26, 2015. (cited on Page 21, 25, 27, 32, 38, and 61)

Ling Qi, Zhuo Tang, Yunchuan Qin, and Yu Ye. CSRA: An Efficient Resource Allocation Algorithm in Mapreduce Considering Data Skewness. In *Proceedings of the International Conference on Knowledge Science, Engineering and Management (KSEM)*, pages 651–662. Springer, 2015. (cited on Page 83)

Kun Qian, Lucian Popa, and Prithviraj Sen. Active Learning for Large-scale Entity Resolution. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, pages 1379–1388, 2017. (cited on Page 111)

Database Group of Prof. Erhard Rahm. Benchmark Datasets for Entity Resolution. Available at: https://dbs.uni-leipzig.de/research/projects/object_matching/benchmark_datasets_for_entity_resolution, 2017. (cited on Page 92, 100, 103, and 105)

Smriti R Ramakrishnan, Garret Swart, and Aleksey Urmanov. Balancing Reducer Skew in Mapreduce Workloads Using Progressive Sampling. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*, page 16. ACM, 2012. (cited on Page 83)

Jason D Rennie, Lawrence Shih, Jaime Teevan, and David R Karger. Tackling the Poor Assumptions of Naive Bayes Text Classifiers. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 616–623, 2003. (cited on Page 11 and 103)

Chuitian Rong, Wei Lu, Xiaoli Wang, Xiaoyong Du, Yueguo Chen, and Anthony KH Tung. Efficient and Scalable Processing of String Similarity Join. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 25(10):2217–2230, 2012. (cited on Page 21, 28, 41, and 42)

Chuitian Rong, Chunbin Lin, Yasin N Silva, Jianguo Wang, Wei Lu, and Xiaoyong Du. Fast and Scalable Distributed Set Similarity Joins for Big Data Analytics. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 1059–1070. IEEE, 2017. (cited on Page 21, 25, 26, 28, 31, 37, and 39)

Chuitian Rong, Xiaohai Cheng, Ziliang Chen, and Na Huo. Similarity Joins for High-dimensional Data Using Spark. *Concurrency and Computation: Practice and Experience*, 31(20):e5339, 2019. (cited on Page 21, 22, 25, 27, 28, 32, 38, 39, 42, and 62)

Semih Salihoglu and Jennifer Widom. GPS: A Graph Processing System. In *Proceedings of the International Conference on Scientific and Statistical Database Management (SSDBM)*, page 22. ACM, 2013. (cited on Page 44)

Walter Santos, Thiago Teixeira, Carla Machado, W Meira, Altigran S Da Silva, DR Ferreira, and Dorgival Guedes. A Scalable Parallel Deduplication Algorithm. In *Proceedings of the Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 79–86. IEEE, 2007. (cited on Page 20, 21, 22, 23, 27, 28, 33, and 44)

Sunita Sarawagi. Special Issue on Data Cleaning. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering (IEEE-CS)*, 23(4):2–3, 2000. (cited on Page 6)

Sunita Sarawagi and Anuradha Bhamidipaty. Interactive Deduplication Using Active Learning. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 269–278. ACM, 2002. (cited on Page 2, 3, 5, 98, 102, 104, 107, 108, 109, 111, and 112)

Michael Schroeck, Rebecca Shockley, Janet Smart, Dolores Romero-Morales, and Peter Tufano. Analytics: The Real-world Use of Big Data. *IBM Global Business Services*, 12(2012):1–20, 2012. (cited on Page 1)

Seung H. Sebastian, Manfred Opper, and Haim Sompolinsky. Query by Committee. In *Proceedings of the Workshop on Computational Learning Theory (COLT)*, 1992. (cited on Page 13)

Toby Segaran. *Programming Collective Intelligence*. O'Reilly Media, 2008. (cited on Page 11)

Thomas Seidl, Sergej Fries, and Brigitte Boden. Mr-dsj: Distance-based Self-join for Large-scale Vector Data Analysis with Mapreduce. In *In Fachtagung Datenbanksysteme für Business, Technologie und Web (BTW)*, volume 214, pages 37–56, 2013. (cited on Page 20, 26, 30, 36, 41, and 42)

Burr Settles. Active Learning Literature Survey. Technical report, University of Wisconsin-Madison, Department of Computer Sciences, 2009. (cited on Page 12)

Mohamed Ahmed Sherif and Axel-Cyrille Ngonga Ngomo. An Optimization Approach for Load Balancing in Parallel Link Discovery. In *Proceedings of the International Conference on Semantic Systems (SEMANTiCS)*, pages 161–168. ACM, 2015. (cited on Page 83)

Yasin N Silva and Jason M Reed. Exploiting Mapreduce-based Similarity Joins. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 693–696. ACM, 2012. (cited on Page 20, 26, 29, 35, and 39)

Yasin N Silva, Jason M Reed, and Lisa M Tsosie. Mapreduce-based Similarity Join for Metric Spaces. In *Proceedings of the International Workshop on Cloud Intelligence (Cloud-I)*, page 3. ACM, 2012. (cited on Page 20 and 39)

Temple F Smith, Michael S Waterman, et al. Identification of Common Molecular Subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981. (cited on Page 9)

Mohammad Karim Sohrabi and Hosseion Azgomi. Parallel Set Similarity Join on Big Data Based on Locality-sensitive Hashing. *Science of Computer Programming*, 145:1–12, 2017. (cited on Page 21, 26, 31, and 37)

Michael Stonebraker, Daniel Abadi, David J DeWitt, Sam Madden, Erik Paulson, Andrew Pavlo, and Alexander Rasin. Mapreduce and Parallel Dbmss: Friends or Foes? *Communications of the ACM*, 53(1):64–71, 2010. (cited on Page 22 and 49)

Xian-He Sun and Lionel M Ni. Scalable Problems and Memory-bounded Speedup. *Journal of Parallel and Distributed Computing*, 19(1):27–37, 1993. (cited on Page 15)

Zhuo Tang, Wen Ma, Kenli Li, and Keqin Li. A Data Skew Oriented Reduce Placement Algorithm Based on Sampling. *IEEE Transactions on Cloud Computing (TCC)*, 2016. (cited on Page 83)

Zhuo Tang, Xiangshen Zhang, Kenli Li, and Keqin Li. An Intermediate Data Placement Algorithm for Load Balancing in Spark Computing Environment. *Future Generation Computer Systems (FGCS)*, 78:287–301, 2018. (cited on Page 83)

Sheila Tejada, Craig A. Knoblock, and Steven Minton. Learning Object Identification Rules for Information Integration. *Information Systems*, pages 607–633, 2001. (cited on Page 3, 98, 107, 108, 109, 111, and 112)

Khoi-Nguyen Tran, Dinusha Vatsalan, and Peter Christen. Geco: An Online Personal Data Generator and Corruptor. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, pages 2473–2476, New York, NY, USA, 2013. ACM. (cited on Page 18)

Dinusha Vatsalan, Dimitrios Karapiperis, and Aris Gkoulalas-Divanis. An Overview of Big Data Issues in Privacy-preserving Record Linkage. In *International Symposium on Algorithmic Aspects of Cloud Computing (ALGOCLOUD)*, pages 118–136. Springer, 2018. (cited on Page 46)

Rares Vernica, Michael J Carey, and Chen Li. Efficient Parallel Set-similarity Joins Using Mapreduce. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 495–506. ACM, 2010. (cited on Page 20, 26, 28, 29, and 35)

Vassilios S Verykios, George V Moustakides, and Mohamed G Elfeky. A Bayesian Decision Model for Cost Optimal Record Matching. *The VLDB Journal*, 12(1): 28–40, 2003. (cited on Page 10)

Chaokun Wang, Jianmin Wang, Xuemin Lin, Wei Wang, Haixun Wang, Hongsong Li, Wanpeng Tian, Jun Xu, and Rui Li. Mapdupreducer: Detecting near Duplicates Over Massive Datasets. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 1119–1122. ACM, 2010. (cited on Page 20, 24, 25, 28, 29, and 35)

Chen Wang and Sarvnaz Karimi. Parallel Duplicate Detection in Adverse Drug Reaction Databases with Spark. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, pages 551–562, 2016. (cited on Page 62)

Qing Wang, Dinusha Vatsalan, and Peter Christen. Efficient Interactive Training Selection for Large-scale Entity Resolution. In *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, pages 562–573, 2015. (cited on Page 98)

Michael S Waterman, Temple F Smith, and William A Beyer. Some Biological Sequence Metrics. *Advances in Mathematics*, 20(3):367–387, 1976. (cited on Page 9)

Steven Euijong Whang, David Menestrina, Georgia Koutrika, Martin Theobald, and Hector Garcia-Molina. Entity Resolution with Iterative Blocking. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 219–232. ACM, 2009. (cited on Page 43)

William E Winkler. Overview of Record Linkage and Current Research Directions. In *Bureau of the Census*. Citeseer, 2006. (cited on Page 45)

Chuan Xiao, Wei Wang, Xuemin Lin, Jeffrey Xu Yu, and Guoren Wang. Efficient Similarity Joins for Near-duplicate Detection. *ACM Transactions on Database Systems (TODS)*, 36(3):15, 2011. (cited on Page 35 and 38)

Reynold Xin, Parviz Deyhim, Ali Ghodsi, Xiangrui Meng, and Matei Zaharia. Graysort on Apache Spark by Databricks. *GraySort Competition*, 2014. (cited on Page 65)

Yujie Xu, Peng Zou, Wenyu Qu, Zhiyang Li, Keqiu Li, and Xiaoli Cui. Sampling-based Partitioning in Mapreduce for Skewed Data. In *Proceedings of the ChinaGrid Annual Conference (CHINAGRID)*, pages 1–8. IEEE, 2012. (cited on Page 83)

Wei Yan, Yuan Xue, and Bradley Malin. Scalable Load Balancing for Mapreduce-based Record Linkage. In *Proceedings of the International Conference on Performance Computing and Communications Conference (IPCCC)*, pages 1–10. IEEE, 2013a. (cited on Page 2, 40, 41, 68, and 84)

Wei Yan, Yuan Xue, and Bradley Malin. Scalable and Robust Key Group Size Estimation for Reducer Load Balancing in Mapreduce. In *Proceedings of the International Conference on Big Data (Big Data)*, pages 156–162. IEEE, 2013b. (cited on Page 83)

Byoungju Yang, Hyun Joon Kim, Junho Shim, Dongjoo Lee, and Sang-goo Lee. Fast and Scalable Vector Similarity Joins with Mapreduce. *Journal of Intelligent Information Systems (JIIS)*, 46(3):473–497, 2016. (cited on Page 21, 26, 30, 37, and 41)

Bengio Yoshua, Ducharme Réjean, Vincent Pascal, and Jauvin Christian. A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, 3:1137–1155, 2003. (cited on Page 10)

Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 2–2. USENIX Association, 2012. (cited on Page 34 and 65)

Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J Franklin, et al. Apache Spark: A Unified Engine for Big Data Processing. *Communications of the ACM*, 59(11):56–65, 2016. (cited on Page 16 and 17)

Cha Zhang and Yunqian Ma. *Ensemble Machine Learning: Methods and Applications*. Springer, 2012. (cited on Page 12)

Chi Zhang, Feifei Li, and Jeffrey Jestes. Efficient Parallel Knn Joins for Large Data in Mapreduce. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, pages 38–49, 2012. (cited on Page 20, 25, 26, 29, 32, and 35)

Xujun Zhao, Jifu Zhang, and Xiao Qin. K NN-DP: Handling Data Skewness in Knn Joins Using Mapreduce. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 29(3):600–613, 2017. (cited on Page 21, 25, 26, 31, 32, and 37)

Paul Zikopoulos, Chris Eaton, et al. *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*. McGraw-Hill Osborne Media, 2011. (cited on Page 1)

Justin Zobel and Philip Dart. Phonetic String Matching: Lessons from Information Retrieval. In *Proceedings of the International Conference on Research and Development in Information Retrieval (SIGIR)*, pages 166–172. ACM, 1996. (cited on Page 9)

Ehrenerklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; verwendete fremde und eigene Quellen sind als solche kenntlich gemacht. Insbesondere habe ich nicht die Hilfe eines kommerziellen Promotionsberaters in Anspruch genommen. Dritte haben von mir weder unmittelbar noch mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen.

Ich habe insbesondere nicht wissentlich:

- Ergebnisse erfunden oder widersprüchliche Ergebnisse verschwiegen,
- statistische Verfahren absichtlich missbraucht, um Daten in ungerechtfertigter Weise zu interpretieren,
- fremde Ergebnisse oder Veröffentlichungen plagiiert,
- fremde Forschungsergebnisse verzerrt wiedergegeben.

Mir ist bekannt, dass Verstöße gegen das Urheberrecht Unterlassungs- und Schadensersatzansprüche des Urhebers sowie eine strafrechtliche Ahndung durch die Strafverfolgungsbehörden begründen kann. Die Arbeit wurde bisher weder im Inland noch im Ausland in gleicher oder ähnlicher Form als Dissertation eingereicht und ist als Ganzes auch noch nicht veröffentlicht.

Magdeburg, den 02.11.2020

Xiao Chen