Open Access

Open Journal of Big Data (OJBD)
Volume 4, Issue 1, 2018

http://www.ronpub.com/ojbd
ISSN 2365-029X

Cloud-Scale Entity Resolution: Current State and Open Challenges

Xiao Chen, Eike Schallehn, Gunter Saake

Institute of Technical and Business Information Systems, Otto-von-Guericke-University of Magdeburg, Universittsplatz 2, Magdeburg, Germany, {xiao.chen, eike, saake}@ovgu.de

Abstract

Entity resolution (ER) is a process to identify records in information systems, which refer to the same real-world entity. Because in the two recent decades the data volume has grown so large, parallel techniques are called upon to satisfy the ER requirements of high performance and scalability. The development of parallel ER has reached a relatively prosperous stage, and has found its way into several applications. In this work, we first comprehensively survey the state of the art of parallel ER approaches. From the comprehensive overview, we then extract the classification criteria of parallel ER, classify and compare these approaches based on these criteria. Finally, we identify open research questions and challenges and discuss potential solutions and further research potentials in this field.

Type of Paper and Keywords

Research Review: *entity resolution, record linkage, parallel computing, parallel entity resolution, deduplication, similarity join, data partitioning, load balancing*

1 Introduction

Within a single information system or across different information systems there may exist different descriptions for one real-world entity. The differences may result from typographical errors, abbreviations, data formatting, etc. However, these errors and inconsistencies can limit the applicability of the data for transactional and analytical purposes, and, accordingly, limit the business value of the data. Therefore, it is necessary to be able to resolve and clarify such different representations. 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 also referred to as data matching [15], record linkage [30], de-duplication [81], reference reconciliation [24], or object identification [42]. More specifically, in the database domain, ER is tightly related to the techniques of similarity joins [52].

Today, ER plays a vital role in diverse areas, not only in traditional applications of census, health care, or national security, but also for Internet-based applications such as social media, online shopping, web searches, business mailing lists, etc. In some domains, like the Web of data, ER is a prerequisite to enable semantic search, interlink descriptions and support deep reasoning [15]. It is also an indispensable step in data cleaning [39] [80], data integration [38], and data warehousing [8]. 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 due to its many applications. From its early days, there have been two general goals: efficiency and effectiveness, which mean

how fast and how accurately an ER task can be solved. In recent years, the rise of the Web and Cloud-based applications has led to several extensions of techniques and algorithms for ER. Data of those applications, depending on specific criteria sometimes also referred to as Big Data, is often semi-structured, comes from diverse domains, and exists on a very large-scale. These three properties make it qualitatively different from traditional data, which brings new challenges to ER and in turn require new techniques or algorithms as solutions.

To be specific, specialized similarity measures are required for semi-structured data, cross-domain techniques are needed to handle data from diverse domains, and parallel techniques are needed to make algorithms not only efficient and effective, but also scalable, in order to be able to deal with the large and often unpredictable scale of the data [85]. Parallel techniques developed in response to the large-scale data challenge are the objects of this survey. For the sake of simplicity, we call ER without using any parallel techniques serial ER, otherwise we call it parallel ER. Despite the relatively prosperous stage of parallel ER, the inevitable future expansion in data volume will require further increases in data processing efficiency. To date, however, no thorough survey of parallel ER has been published. Most of the earlier surveys only addressed the general field of ER.

Gu et al. [37] in 2003 described a record linkage system design and summarized common techniques used in each key system component, such as blocking, comparison, decision model, etc. In addition, they also provided new alternatives to implement these components and compared them to previous algorithms. Brizan and Tansel [8] in 2006 summarized matching techniques and ways to apply them. In [90] Winkler described in 2006 various ER applications and then described methods for improving matching efficacy, i.e. to make ER more easily applicable, and according methods for string comparison. Elmagarmid et al. [28] in 2007 gave a thorough analysis of approaches on duplicate record detection, which specifically includes techniques used to match records with single attribute or multiple attributes, techniques for improving the efficiency and scalability of approximate duplicate detection algorithms, a few commercial tools used in industry and a brief discussion of open problems.

Köpcke and Rahm [59] in 2010 compared and evaluated 11 proposed frameworks for ER. The selected comparison criteria can also be used to compare other frameworks. After the theoretical comparison of 11 frameworks, experimental evaluations were also provided to assess the effectiveness and efficiency of frameworks. Christen [16] in 2012 focused on indexing techniques used in ER and gave a detailed discussion of six indexing techniques with a total of twelve variations of them, which included a theoretical analysis of their complexity and an empirical evaluation of these techniques. Getoor und Machanavajjhala [34] [35] in 2012 and 2013 gave tutorials on ER, including existing solutions, current challenges and open questions in the field of ER. Gal [33] in 2014 also gave a tutorial on Entity Resolution, and he concluded models and algorithms used for uncertainty in ER and exposed current challenges and future research directions. Chen [12] published a short survey in 2015 in the area of crowdsourcing ER. Papadakis et al. [74] in 2016 focused also on blocking (also called indexing) techniques used in ER, which has the same focus as [16]. 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.

As outlined above, a survey with a focus on aspects of the parallelization of ER does not yet exist. In this paper we present such a survey, covering the basic principles and most important parallel ER approaches. The main contributions of this work are as follows:

- Extracting the classification criteria of parallel ER approaches.
- Classifying and comparing the important parallel ER approaches based on these criteria.
- Identifying possible solutions that fulfill the important criteria of parallel ER.
- Discussing further research potentials for parallel ER.

The remainder of this paper is structured as follows: In Section 2, we introduce the process of ER and parallel ER, which are the foundations to understand the paper. Then, we present the methodology of our literature review in Section 3. In Section 4, we first group 34 selected approaches into three categories. Based on these three categories, we then provide an overview of the publications and introduce the three sets of criteria we extracted. Lastly, we discuss those publications based on the first two sets of criteria. The discussion based on the last set of criteria, efficiency-based criteria, are presented separately in Section 5 because of their great importance for parallel ER. In this section, we also propose possible solutions for the key criteria and these solutions are inspired and extracted from existing research on parallel ER. In Section 6, we discuss the remaining research potential for parallel ER. Last, we sum up the paper and provide an outlook of possible future work in Section 7.

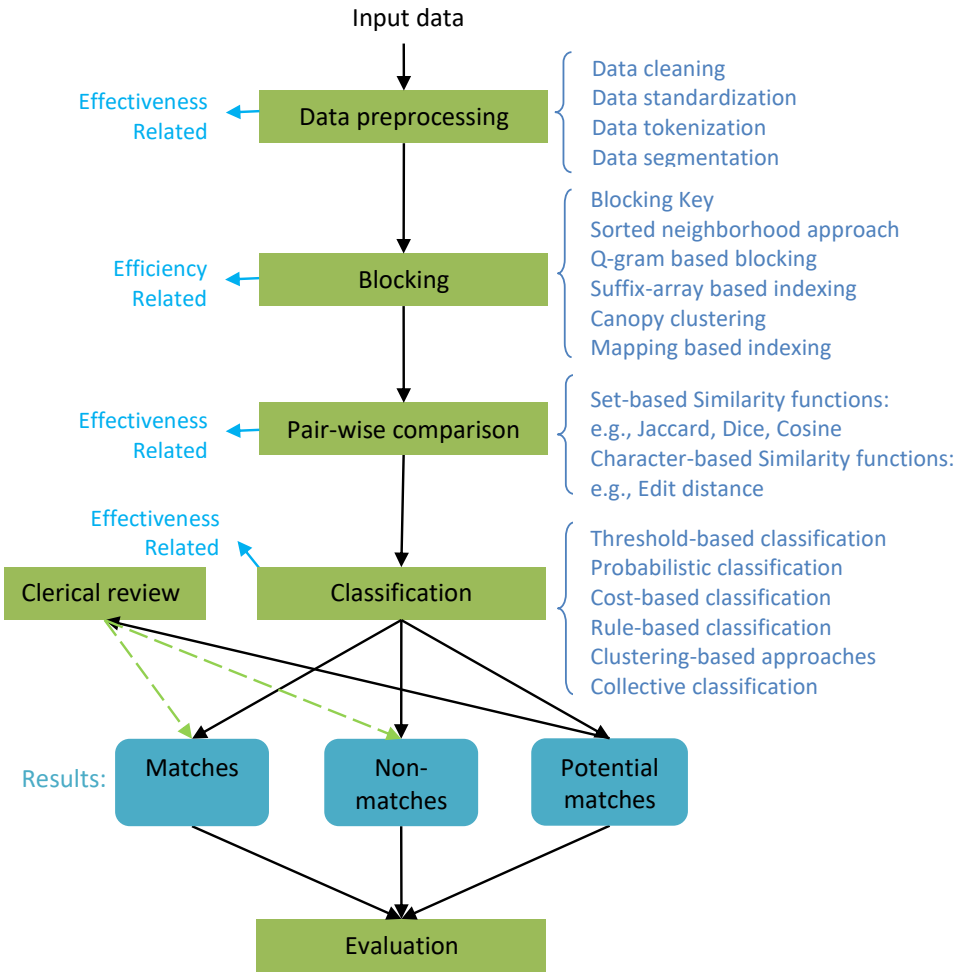


Figure 1: The general process of ER based on [15]

2 PARALLEL ER

In this section, we first introduce the goals that ER solutions should pursue. Then we describe the typical workflow for ER, which is a preliminary to explain the two kinds of parallelism in parallel ER to be introduced at last.

There are three general goals for solving ER tasks: effectiveness, efficiency, and scalability. Effectiveness means how reliable the ER results are, which is often expressed with three metrics: precision, recall and F-measure [64]. Efficiency and scalability are closely related. Efficiency, typically referring to runtime efficiency, indicates a time to complete an ER task that is optimal or sufficient to fulfil application requirements. Scalability indicates the capability of a system to process changing data volumes with the same response time, for instance by flexibly adjusting processing resources.

2.1 General ER Workflow

In order to achieve these goals, there is a typical workflow of ER tasks which is shown in Figure 1. The workflow consists of five major steps in total: data preprocessing, blocking, pair-wise comparison, classification, and evaluation [15]. In the following, these five steps are introduced according to the order of their application in the overall process.

Data preprocessing: The input data that needs to be resolved is usually noisy, inconsistent, and with low data quality. Furthermore, some of the 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 [15].

Blocking: To achieve efficiency in parallel as well as in serial ER processing, blocking is often adopted. Blocking means splitting the whole input dataset into blocks, and then comparing only entities in the same block. This requires a blocking strategy, which guarantees – or at least makes it very likely – that potentially matching records will be assigned to the same block. Though this may have a negative impact on effectiveness, the reason for adopting blocking is efficiency of ER, which in general is a very time-consuming, 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 the computing complexity for it with n records is $O(n^2)$.

Therefore, the computing time for solving an ER task is dramatically extended, when the number of records increases. However, by using blocking, the search space for ER can be reduced and the corresponding computing complexity becomes $O(n)$ for a fixed block size. Furthermore, for parallel processing blocking provides a set of independent sub-tasks that can be easily parallelised. Because of their importance for efficiency, many blocking techniques have been developed. They can be categorized into six types: blocking key (also frequently referred to as standard blocking), sorted neighborhood approach [38], q-gram or n-gram (fixed size sub-strings, overlapping or not overlapping, respectively) based blocking [72], suffix-array based indexing [2], canopy clustering [67] and mapping-based indexing [47].

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 two entities of a pair are. Most often, input entities are considered to be single strings or sequences of strings in records or objects. Therefore, here we focus on strings as input for the similarity functions. There are two main types of string similarity functions. One type is set-based functions, such as Jaccard [44] [23], dice [22] and Cosine [21] similarity functions. The input for set-based similarity functions must be a set of tokens, where a token can be either a word or n-gram/q-gram [72] and can be generated from data preprocessing steps. The other one is character-based, such as edit distance [60] and various derivatives, which are defined based on the number of character operations required to transform one string to another [20].

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: simple threshold-based classification, probabilistic classification, cost-based classification, rule-based classification, clustering-based approaches and collective classification, etc [15]. When some pairs cannot be identified as match or non-match automatically, it is also possible that they are reviewed and a decision is made by a human expert [15].

Evaluation: The last step evaluation is optional. It is used to estimate the effectiveness of the ER process.

Techniques used in data preprocessing, pair-wise comparison, and classification should be chosen carefully to achieve a high effectiveness. Based on the techniques determined in the above steps, blocking techniques are typically used to improve efficiency and scalability, but may imply a trade-off situation with effectiveness goals.

2.2 Workflow of Parallel ER

The typical workflow of parallel ER approaches also follows these five steps. What makes parallel ER different from serial ER is that in some or all steps of the ER process parallel techniques are used to speed-up ER and/or achieve scalability. There are two kinds of possible parallelisations for ER: intra-step and inter-step parallelism.

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 are carried out simultaneously on elements of one data set. In this way, the large-scale data can be divided into many 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 in each step of ER 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.

To date, almost all publications on parallel ER focus on data parallelism. Only [79] presents an approach to parallelise the processing within as well as across different steps.

3 Methodology and Result of Literature Search

In this section, we describe the process of finding and selecting the 44 papers on parallel ER presented in this review. The process includes the following steps and according results:

Identification of a start set of articles: The first task in the literature search is to identify a start set of articles having a strong focus on the considered field of parallel ER. If a start set is too large, the efforts to identify related papers will be unacceptable, while, on the other hand, if a start set contains too few articles, some important articles may be missed. As ER has many synonyms, the most commonly used aliases were also used as key words to achieve a start set covering diverse entry points to sub-fields of research. Furthermore, ER-related research in computer science has been going on for almost six decades, so the number of articles is huge. Therefore, it was mandatory to use key words distinguishing serial and parallel ER to limit the size of the start set. To make the search string more restrictive we use the keyword 'partition'in oder to find the papers with a focus on parallel processing. 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 five big literature databases: ACM Digital Library, IEEE Xplore, SpringerLink, ScienceDirect and Scopus. Returned articles from workshops, conferences and journals of computer science formed the start set, including 9 articles from ACM Digital Library, 439 articles from IEEE Xplore, 249 articles from SpringerLink, 248 articles from SpringerLink and 205 articles from Scopus.

Applying inclusion and exclusion criteria on the start set: The second step is to define inclusion and exclusion criteria and then apply them on the papers in the start set. Inclusion and exclusion criteria are as follows.

Inclusion Criteria:

IC01: Including articles, which are published between 2005 and April 2017. 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 2017 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 lecture notes or summaries of conferences, etc.

EC03: Removing articles, which are literature reviews and surveys.

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 applications domains, such as geospatial, forensic, networking, multimedia domains.

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

EC07: 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 2017, i.e., $2 * (2018 - publicationYear)$.

Table 1: Classification of parallel RE approaches based on the programming model

ID	Publication Information	Programming Model
P-Swoosh [49]	by Kawai et al. in 2006	parallel DBMS
Parallel linkage [50]	by Kim and Lee in 2007	parallel DBMS
D-Swoosh [5]	by Benjelloun et al. in 2007	parallel DBMS
FERAPARDAF [79]	by Santos et al. in 2007	parallel DBMS
Febrl [14]	by Christen in 2008	parallel DBMS
Iterative DDG [40]	by Herschel et al. in 2012	parallel DBMS
Partition-based [46]	by Jiang et al. in 2013	parallel DBMS
Pairwise document [29]	by Elsayed et al. in 2008	MapReduce
SSJ-2R [3]	by Baraglia et al. in 2010	MapReduce
Dedoop [51–58]	by Kirsten et al. from 2010 to 2013	MapReduce
VCL [87]	by Vernica et al. in 2010	MapReduce
MapDupReducer [88]	by Wang et al. in 2010	MapReduce
MD-Approach [18]	by Dal Bianco et al. in 2011	MapReduce
V-SMART-Join [71]	by Metawally et al. in 2012	MapReduce
MRSimJoin [83, 84]	by Silva et al. in 2012	MapReduce
LINDA [7]	by Boehm et al. in 2012	MapReduce
Graph-based [48]	by Kardes et al. in 2013	MapReduce
MR-DSJ [82]	by Seidl et al. in 2013	MapReduce
PHiDJ [32]	by Fries et al. in 2014	MapReduce
Cluster Join [19]	by Das Sarma et al. in 2014	MapReduce
Graph-parallel [66]	by Malhotra et al. in 2014	MapReduce
Mass Join [20]	by Deng et al. in 2014	MapReduce
DCS++ MR-ER [69] [36]	by Mestre et al. in 2013 and 2015	MapReduce
Sort-Map-Reduce [62] [63]	by Ma et al. in 2015	MapReduce
FACET [93]	by Yang et al. in 2015	MapReduce
SJT-based [61]	by Liu et al. in 2016	MapReduce
High velocity streams [6]	by Benny et al. in 2016	MapReduce
Dis-Dedup [17]	by Chu et al. in 2016	MapReduce
Meta-blocking [27]	by Efthymiou et al. in 2017	MapReduce
Sampling-based [11]	by Chen et al. in 2017	MapReduce
Density-based blocking [25]	by Dou et al. in 2017	MapReduce or Spark RDD
RDD-based ER [10]	by Chen et al. in 2015	Spark RDD
ER of healthcare data [76]	by Pita et al. in 2015	Spark RDD
DCS++ Spark-ER [70]	by Demetrio et al. in 2017	Spark RDD

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 from article contents, i.e., IC01, IC02, and EC01, applying the filter functions of each search engine of the literature databases. After this sub-step, 1119 articles remained.

Sub-step 2: Apply the criteria (IC03-IC04 and EC02-EC06) on article titles to include or remove articles. After this substep, 118 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, 31 articles are left. And lastly by applying EC07, eight articles are removed because of not reaching the citation lower-bound. After this sub-step, 23 articles are left.

Sub-step 4: Apply the criteria (IC03-IC04 and EC01-EC06) 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 20 articles are left.

Supplementing the literature set by checking the references of articles: After we had the list of identified

20 articles from the second step, we checked references from or to those articles, and then decided whether to add found articles to our literature set or not based on all criteria described above. After this step, we found 24 more articles and had 44 articles in total.

4 Overview and Classification

Table 1 lists the 44 papers selected. 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 34 rows in Table 1.

In this section, we first classify all approaches according to the programming model they used into three categories: Parallel DBMS, i.e., parallelism is achieved in terms of parallel database processing and no specific programming model, MapReduce-based, and 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 programming models and frameworks are a topic of ongoing research and their applicability and usage for ER must be covered in future research, as discussed in section 6.

Then we extract three sets of criteria for parallel ER to provide an overview for existing approaches based on these three categories. The reason for this is, that the used programming model is a basic aspect of parallel ER. All further discussions are based on these three categories. The three sets of criteria we extracted are as follows.

- 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 preprocessing, similarity function, match mode and clustering.
- The last set of criteria pertains 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 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 applications. In the rest of this section, we will introduce the first two sets of criteria and present the classification based on those directly after each set of criteria. Efficiency-based criteria are discussed in the next section separately, because efficiency-based criteria, including considerations related to scalability, are the most critical ones for parallel ER.

4.1 Classification of Approaches Based on Frameworks Used

In this survey, we present 34 approaches of parallel ER. For simplicity, we assign each publication a short and meaningful identification to simply represent them throughout this survey. Then we classify them according to the programming model: parallel DBMS, i.e., no programming model used, MapReduce-based and Spark-based. Table 1 shows all publications and their programming model. As we can see, seven of them do not apply a programming model but use Parallel DBMS to execute parallel ER tasks. 23 approaches use only MapReduce, and the remaining 3 approaches are only Spark-based. The approach presented in [25] implemented parallel ER until finishing its blocking step with both MapReduce and Spark.

More than two thirds of the approaches chose MapReduce to implement parallelism, and only about one fifth of the research implemented general parallelism, i.e., using Parallel DBMS, and 4 out of 34 approaches are Spark-based, 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-sourced Top-Level Apache Project since February 2014[1]. The research not using any specific programming model was mostly early work, but it is certainly more relevant for real-world applications than the proportion of academic research suggests. After MapReduce became popular for parallel computing and with the support of its open-sourced implementation of Apache Hadoop, the vast majority of research approaches after 2008 applied it. The reason is that MapReduce provides users a model to simply express relatively sophisticated distributed programs [75]. 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.

As mentioned above, almost all approaches only parallelised the pair-wise-comparison step, while [79] parallelised data processing in each step and also

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

Table 2: Classification based on the general criteria, grouped by programming models

(a) General classification of parallel DBMS ER

ID	Operation Type	#Input Sources	Input Data Type
P-Swoosh [49]	entity resolution	not described	records
Parallel linkage [50]	entity resolution	2	records
D-Swoosh [5]	entity resolution	not described	records
FERAPARDAF [79]	deduplication	1	records
Febrl [14]	deduplication	1	records
Iterative DDG [40]	deduplication	1	graph
Partition-based [46]	similarity join	1	strings

(b) General classification of MapReduce-based ER

ID	Operation Type	#Input Sources	Input Data Type
Pairwise document	similarity self-join	1	documents
SSJ-2R [3]	similarity self-join	1	documents
Dedoop [51–58]	entity resolution	1 or n	records
VCL [87]	set-similarity join	1 or n	records
MapDupReducer [88]	deduplication	1	documents
MD-Approach [18]	deduplication	1	records
V-SMART-Join [71]	similarity join	1	sets, multisets, and vectors
MRSimJoin [83, 84]	similarity join	2	records
LINDA [7]	entity resolution	n	graph
Graph-based [48]	entity resolution	not described	records
MR-DSJ [82]	similarity self-join	1	vector data
PHiDJ [32]	similarity self-join	1	high dimensional vectors
Cluster Join [19]	similarity join	1 or n	records
Graph-parallel [66]	entity resolution	not described	records
Mass Join [20]	similarity join	1 or 2	strings
DCS++ MR-ER [69] [36]	entity resolution	1	records
Sort-Map-Reduce [62] [63]	deduplication	1	records
FACET [93]	similarity join	1 or 2	vectors
SJT-based [61]	similarity join	1	records
High velocity streams [6]	entity resolution	2	records
Dis-Dedup [17]	deduplication	1	records
Meta-blocking [27]	entity resolution	2	records
Sampling-based [11]	similarity join	2	records
Density-based blocking [25]	entity resolution	1 or 2	records

(c) General classification of Spark-based ER

ID	Operation Type	#Input Sources	Input Data Type
Density-based blocking [25]	entity resolution	1 or 2	records
RDD-based ER [10]	top-k similarity join	1	multi-dimensional
ER of healthcare data [76]	entity resolution	n	health data
DCS++ Spark-ER [70]	entity resolution	1 or 2	records

performed different steps in parallel through its run-time system, Anthill. In addition, among publications that implement data parallelism, most of them considered only inter-processor parallelism. Approaches presented in [46], [18], [7] and [66] considered both intra and inter-processor parallelism. The benefit of considering intra-processor parallelism was mentioned in [18], i.e. the communication overhead is low without excessive data copying.

Using parallel DBMS has some shortcomings for small and medium-sized enterprises. Parallel DBMSs are expensive [86], have no ability to operate in a heterogeneous environment and have very limited fault tolerance [1]. Furthermore, they require high maintenance and administration efforts. In academia, researchers may encounter problems, such as limited budget and a heterogeneous environment. This may be another reason, why open-source frameworks are unproportionally popular there. Furthermore, their high level of abstraction might help researchers to concentrate on application specific – in this case ER – research questions.

4.2 General Criteria and 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:

Operation Type: As mentioned above, de-duplication 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 with ER, normally they have their particular emphasises. 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 whether algorithms or techniques used support ER only within one single source or also among multiple sources. 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 actually 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.

Input Data Type: For existing publications, the type of input data varies. In most cases, approaches do not limit the types of the input data. We use "records" to represent this case, because using flatly structured data units as in relational databases are 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, in order 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 2a, Table 2b, and Table 2c, which provide an overview for these criteria, once again grouped for parallel DBMS ER (Table 2a), MapReduce-based ER (Table 2b) and Spark-based ER (Table 2c). As can be seen in these three overview tables regarding the **operation type**, the majority of publications considered a general entity resolution problem. Some of them consider only the situation that there is a single input source, which indicates a de-duplication problem. Different kinds of similarity joins are also addressed, such as set or top-k similarity joins.

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 [50], 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**. Three of them only accept input data with strings and one of them tried to resolve document matches. RDD-based ER [10] and ER of health-care data [76] focus on multi-dimensional and health data due to their specific application area. Furthermore, Iterative DDG [40] and LINDA [7] 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: Effectiveness consideration of parallel DBMS ER

ID	Data Preprocessing	Similarity Function	Match Mode	Clustering
P-Swoosh [49]	no preprocessing	not described	merge after match	no clustering
Parallel linkage [50]	no preprocessing	s-cc/s-dcself/s-dd1/ s-dd2/s-dd3	merge after match	no clustering
D-Swoosh [5]	no preprocessing	not described	merge after match	no clustering
FERAPARDAF [79]	standardization, cleaning	not described	no merge	no clustering
Febrl [14]	hidden-markov-model for cleaning/standardization	not described, matching attribute weights	no merge	no clustering
Iterative DDG [40]	entity graph as input	self-defined similarities	iterative and propagate	no clustering
Partition-based [46]	partition and substring comparison	extension-based verification	no merge	no clustering

Table 4: Effectiveness consideration of Spark-based ER

ID	Data Preprocessing	Similarity Function	Match Mode	Clustering
Density-based blocking [25]	no preprocessing	none, focus only on blocking algorithms	no merge	no clustering
RDD-based ER [10]	no preprocessing	top-k-DC	no merge	k closest pairs
ER of healthcare data [76]	no preprocessing	dice or bit vectors comparison	no merge	no clustering
DCS++ Spark-ER [70]	no preprocessing	Jaro-Winkler	no merge	no clustering

4.3 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.

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, Table 4 and Table 5 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 [14], [79] and [88] also supported data cleaning and standardization. In [14] Christen et al. suggested a three-step data cleaning and standardization based on hidden Markov models. As a preparation to special blocking methods used in some publications, the data preprocessing step in [88], [20], [87] and [77] include transforming input records to tokens, and [46] first partitions the input records (records here are all strings) and then transforms input strings into substrings. The

Table 5: Effectiveness consideration of MapReduce-based ER

ID	Data Preprocessing	Similarity Function	Match Mode	Clustering
Pairwise document [29]	Stemming & stopwords removal & df-cut	symmetric variant of OkapiBM25 [73]	no merge	no clustering
SSJ-2R [3]	stemming; normalization; stopwords removal; lexicon extraction; sorting features	not described	no merge	no clustering
Dedoop [51–58]	no preprocessing	not described	no merge	no clustering
VCL [87]	string to prefix tokens	Coefficients: Jaccard, Tanimoto, cosine	no merge	no clustering
MapDup-Reducer [88]	cleaning, parsing, tokensation	not described	no merge	no clustering
MD-Approach [18]	no preprocessing	not described	no merge	no clustering
V-SMART-Join [71]	stopwords removal	Nominal Similarity Measures	no merge	no clustering
MRSimJoin [83, 84]	no preprocessing	various functions possible	no merge	no clustering
LINDA [7]	edges added to build graph	not described	iterative and propagate	no clustering
Graph-based [48]	Soundex or Phoetex	feature-based	no merge	transitive closure & sClust
MR-DSJ [82]	no preprocessing	distance-based functions	no merge	no clustering
PHiDJ [32]	no preprocessing	distance-based functions	no merge	no clustering
Cluster Join [19]	no preprocessing	not described	no merge	no clustering
Graph-parallel [66]	no preprocessing	not described	merge or no merge	connected components
Mass Join [20]	token-based signature generation	Jaccard, edit distance, set- and character-based	no merge	no clustering
DCS++ MR-ER [69] [36]	no preprocessing	Jaro-Winkler	no merge	no clustering
Sort-Map-Reduce [62] [63]	no preprocessing	not described	no merge	no clustering
FACET [93]	getting additional information to prepare for blocking	cosine or Dice similarity(-based)	no merge	no clustering
SJT-based [61]	no	not described	no merge	no clustering
High velocity streams [6]	2ord boundaries, stemming, stop word removal	average result of 13 functions	no merge	no clustering
Dis-Dedup [17]	no preprocessing	edit distance	no merge	no clustering
Meta-blocking [27]	redundancy positive block collections	not described	no merge	no clustering
Sampling-based [11]	centroid selection based on sampled data	any functions possible	no merge	no clustering
Density-based blocking [25]	no preprocessing	none, focus only on blocking algorithms	no merge	no clustering

approaches presented in [7] and [40] are graph-based and consider relationships between input records. Therefore, they build an entity graph during the data preprocessing step. Besides, in [48], Soundex or Phoetex is used to preprocess the input data.

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, and around half of the approaches did not discuss the specific similarity function(s) they used. Dou et al. [25] focused their research only on blocking algorithms and in their research no related information on later steps is described. Kim and Lee [50] developed a series of algorithms for ER between two sources by considering different scenarios whether sources are clean or dirty. 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.

Regarding the **match mode**, most of the approaches terminate ER tasks after they have matching results for all records. In [40] and [7], 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 [66] Malhotra took each connected component as a cluster. Kardes in [48] proposed a clustering strategy called sClust to better cluster records based on the results after computing the transitive closure. The cluster approach used in [10] is to take the top-k closest pairs of records as a cluster.

5 Efficiency-related Criteria and Their Classification

In this section, we present 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 parallelised 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: Blocking 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.

Data partitioning: How to partition the input data or data after defining blocking keys and allocate it to available cores or processors is an important research question in parallel ER. Even if many cores or processors are available for ER, if the partition is not balanced, small data sets lead to idle cores or processors, and large data sets lead to a long processing time of assigned cores or processors, which can dominate the whole run time and lower the performance. Therefore, suitable data partitioning strategies are required.

Load balancing: This criterion is tightly related to the last criterion: data partitioning. But even if the data has been partitioned evenly to avoid data skew, the work load may still suffer from processing skew.

Redundancy handling: Redundancy handling 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 efforts between different processors.

Table 6, Table 7, Table 8 and Table 9 provide an overview and classification of the 34 considered approaches based on the above-explained efficiency criteria. Since data partitioning and load balancing is tightly related, they are in the tables as one column. Because efficiency-related criteria, as a motivation of parallel ER, are 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 subsections.

5.1 Blocking

Blocking as an efficient technique to reduce the search space is considered by most publications. Standard blocking, i.e. using single attributes or combined/ concatenated attributes as blocking keys, 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 [38], q-grams [72], inverted indexes [4] and locality sensitive hashing [43], 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 [91].

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

Table 6: Efficiency consideration of parallel DBMS ER

ID	Blocking	Data partitioning & Load balancing	Redundancy handling
P-Swoosh [49]	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 [50]	not described	replication of source A to all processors and evenly partitioned source B	not described
D-Swoosh [5]	no or standard blocking	scope functions	value equality, hierarchies, linear ordering & Reps functions
FERAPA-RDAF [79]	standard blocking	a labeled stream in Anthill	not described
Febrl [14]	standard blocking, sorted neighborhood, q-gram	not described	not described
Iterative DDG [40]	entity graph as input	evenly partition input entity graph	not described
Partition-based [46]	inverted index	evenly partitioning	substring selection; content filter; effective indexing

as a method to solve the load balancing problem in the following.

In this paper, we only present an overview and choices made in existing publications for the blocking step in parallel ER, and do not discuss the details and the performance issues for different blocking techniques, as mentioned above, Christen and Papadakis et al. provided more detailed discussions and evaluations for existing blocking techniques [74] [16].

5.2 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.

However, data partitioning and load balancing are important for parallel ER and may significantly influence the performance. According to [51] the following three general types of data 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 three general strategies to partition the data to each node, Silva et al. provided a partitioning method from a special perspective in [83,84] by extending the QuickJoin ball partitioning [45] to partition the input dataset iteratively until the sizes of all partitions of data fit a single node, and similarity comparison is only needed to be done within a single node. This makes the QuickJoin ball partitioning become its blocking technique at the same time. Accordingly, their partitioning method also belongs to the above-concluded block-based partitioning.

5.3 Load Balancing

As can be seen in the relevant tables, only few of the presented approaches proposed a specific load balancing strategy, but rather focused on other implementation aspects. However, load balancing is very important and is a factor that can significantly influence the efficiency of parallel ER. If the workload is not evenly assigned to available nodes, the nodes with a heavier load will

Table 7: Efficiency consideration of mapReduce-based ER (1)

ID	Blocking	Data partitioning & Load balancing	Redundancy handling
Pairwise document [29]	inverted index	block-based	not described
SSJ-2R [3]	prefix filtering with inverted index	block-based & bucketing technique	broadcast the remainder file
Dedoop [51–58]	standard blocking & (multi-pass) sorted neighborhood	BlockSplit & PairRange (BDM)	Check overlapping blocking keys
VCL [87]	prefix tokens or PP-join+ [91]	3 stages (BTO/OPTO; BK/PK; BRJ/OPRJ) & a Round-Robin order	not described
MapDup-Reducer [88]	PP-join+ [91]	block-based	not described
MD-Approach [18]	two-step blocking with sliding windows functions	block-based	not described
V-SMART-Join [71]	virtual inverted index	stopword removal or dividing overloaded reducer (sharding algorithm); MapReduce combiner	not described
MRSimJoin [83, 84]	ball partitioning in QuickJoin [45]	base and window-pair parition	not described
LINDA [7]	first rank pairs to assign similar pairs to a same workpackage	workpackage-based & server-controlled	not described
Graph-based [48]	two-step blocking with binomial tree structure	block-based	not described
MR-DSJ [82]	grid-based blocking	block-based	smaller or equal cell ID & bit code & MindistCell & MindistPair
PHiDJ [32]	grid-based blocking	block-based & variable grid width	all measures in MR-DSJ & dimension group ID

dominate the runtime and lower the response time. Therefore, the problem of unbalanced workloads has to be taken seriously.

Some approaches proposed one or more solutions to solve problems of unbalanced load of parallel ER. Based on the publications referenced in the tables, as well as other load-balancing-focused publications, we point out two typical solutions.

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. In addition, in the presented approaches two minor techniques are used to optimize this method. One suggests using a binomial tree structure to conduct (sub-)blocks [68], the other one is using a sliding window to lower the false negative rate [18].

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 all blocks in a similar size and then redivided blocks, also called partitions, are assigned to each processing unit [56]. This solution appears to be similar to prevention-based methods. However,

Table 8: Efficiency Consideration of MapReduce-based ER (2)

ID	Blocking	Data partitioning & Load balancing	Redundancy handling
Cluster Join [19]	each partition as one block	home and outer partitioning & dynamic, 2d-hashing to split oversized partitions	candidate filters;remove mapping-phase-redundancy
Graph-parallel [66]	locality sensitive hashing via Min-hash	first loading a graph to show the blocking result, then sending record to buckets with vertexes & RCP	first transferring record ID then real records
Mass Join [20]	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 [69] [36]	DCS++ [26]	BlockSlicer	transitive closure
Sort-Map-Reduce [62] [63]	(multi-pass) sorted neighborhood	partition using preset functions	not described
FACET [93]	prefix and length filtering	not described	Removing duplicate pairs using key
SJT-based [61]	SJT indexing	extended EFM graph partitioning	inter-node comparison pruning
High velocity streams [6]	weighted-graph-based blocking	pair-based	pruning graph of blocking
Dis-Dedup [17]	Min-hash	triangle distribution	avoid comparing redundant pairs
Meta-blocking [27]	three-stage Meta-blocking	exploiting the power law distribution of block carnality then evenly partitioning	not described
Sampling-based [11]	each partition is a block	CPM and KPM partition methods to achieve load balancing	range-object, double-pivot, pivot filtering, and plane sweeping techniques
Density-based blocking [25]	density-based blocking	randomly split the dataset	not described

Table 9: Efficiency consideration of Spark-based ER

ID	Blocking	Data partitioning & Load balancing	Redundancy handling
Density-based blocking [25]	density-based unsupervised blocking	randomly spit the dataset	not described
RDD-based ER [10]	locality sensitive hashing & BKDRhash function	block-based	Spark filter
ER of healthcare data [76]	standard blocking	block-based	not described
DCS++ Spark-ER [70]	DCS++ [26]	fixed input partition size	transitive closure

Table 10: Classification of load balancing techniques

First Step	Second Step	Proposed approaches
Block Distribution Matrix	Block-based	BlockSplit [56]
		BlockSlicer [36]
	Pair-based	PairRange [56]
Sketch-based data profiling	Block-based	Cell-block devision [92]
	Pair-based	Cell-range devision [92]

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, two basic data structures are proposed to store the block size information: one is a matrix [56], the other one is to adopt a FastAGMS sketch [92] to estimate the block size, which is more scalable than the matrix [92]. For the second step, oversized block elimination can be a block-based or a pair-based approach [56]. Block-based approach means to directly divide the oversized blocks into smaller ones. Pair-based approach means calculating all needed compared pairs and evenly distribute them to nodes. Compared to the block-based approach, the pair-based approach can generate a more balanced workload but its additional overhead will deteriorate the overall execution time when the data set is relatively small [56]. Different approaches have been developed and a short overview is provided in Table 10. Therein, both BlockSplit and BlockSlicer use a block distribution matrix to store block sizes and then divide oversized blocks into small sub-blocks. The difference is that BlockSlicer generates less key-value pairs for the reduce phase and the performance is improved [36]. The other solution is to generate all pairs based on the result of blocking and assign each pair a number to count and mark each record pair, then sending them to the processing unit by round robin [41].

5.4 Redundancy Handling

In the four tables on efficiency (Table 6, Table 7, Table 8, and Table 9), 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 to 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:

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: [5], [19], [20], [55], [57], [93], [82] and [32]. 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.

Transitive closure: Transitive closure means that a record pair can be directly considered as match or non-match without a comparison between them, if they can be deduced with the following two rules. The first one is the deduced match case: If we know record pairs (a, b) and (b, c) are both match pairs, then we deduce that the pair (a, c) is also a match pair. The second one is the deduced non-match case: If we know the record pair (a, b) is a match pair and the other record pair (b, c) is a non-match pair, then we deduce that the pair (a, c) is a non-match pair. Mestre et al. have applied the transitive closure during the step of pair-wise comparison to reduce the number of record pairs that need to be compared [70] [69] [36].

Further pruning techiques: There is a variety of other pruning technologies used to reduce the number of record pairs that address different aspects of the input data. The details of these pruning technologies and processing frameworks can be found in [20], [19], [27], [6], [46], [11], [61], [82] and [32].

Avoiding the transfer of unnecessary data: All of the above three categories are used to reduce the number of required comparisons. However, redundancy may also refer to the communication efforts between different processors or nodes for transferring unnecessary data. Malhotra et al. [66] and Deng et al. [20] presented their measures to avoid unnecessary communication cost, which transfer record IDs instead of records themselves to reduce the overhead. Baraglia et al. [3] used

broadcast to reduce the communication overhead.

This list of types does indicate further optimisation potential for existing approaches and may serve as a guideline to consider during the development of new approaches.

6 Open Challenges

From the descriptions in the previous sections, it is obvious that currently there are a number of 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, a number of basic questions remain open for future solutions in parallel ER. These open challenges are:

Choosing a suitable big data processing framework: As outlined before, there was no systematic analysis of required properties of a framework. MapReduce was mostly used, because it was popular and allowed some improvement. So far, there was only very limited research on Spark-based ER. Spark 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. Besides Hadoop MapReduce and Apache Spark, there have been other new frameworks developed in recent years, e.g., Apache Flink [9] [31]. However, which framework is the best option depends on the specific application scenarios.

Comparison between different implementations: Although MapReduce-based parallelism is very popular, there are still debates on its performance and other aspects. In some papers, its performance is proven to be worse than general parallel programming for some data processing tasks [86] [75]. Therefore, one open issue is to compare the MapReduce-based parallel and general parallel data processing comprehensively for solving ER.

Blocking techniques for large-scale data: Some familiar blocking techniques such as canopy clustering [67], iterative blocking [89], are not deeply studied in parallel ER and new blocking techniques may be developed for large-scale data.

Graph-based parallel ER: Currently there is only little research on this area. 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 [78], Pregel [65], and Giraph [13].

Task parallelism: No research except [79] discusses task parallelism in ER . However, since each step in ER 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.

7 Conclusions and Future Work

In this paper we conducted a comprehensive survey on parallel ER approaches and identify their classification 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 effective, including data preprocessing, similarity function, match mode and clustering.

Efficiency-based criteria are the most important ones for parallel ER, which pursues higher efficiency. This set of criteria 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 key 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?

Important open issues in the area of parallel ER are discussed. Accordingly, our future work will be focused on architectures and frameworks, starting by investigating Spark-based parallel ER methods and its evaluation in comparison to others.

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

ACKNOWLEDGMENTS

We would like to thank China Scholarship Council (CSC) to fund our work, and we are also very grateful to Gabriel Campero Durand for his valuable feedback.

REFERENCES

[1] A. Abouzeid, K. Bajda-Pawlikowski, D. Abadi, A. Silberschatz, and A. Rasin, "Hadoopdb: an architectural hybrid of mapreduce and dbms technologies for analytical workloads," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 922–933, 2009.

[2] A. Aizawa and K. Oyama, "A fast linkage detection scheme for multi-source information integration," in *Web Information Retrieval and Integration, 2005. WIRI'05. Proceedings. International Workshop on Challenges in*. IEEE, 2005, pp. 30–39.

[3] R. Baraglia, G. De Francisci Morales, and C. Lucchese, "Document similarity self-join with mapreduce," in *Data Mining (ICDM), 2010 IEEE 10th International Conference on*. IEEE, 2010, pp. 731–736.

[4] C. Bell, I. H. Witten, and A. Moffat, "Managing gigabytes," *Compressing and Indexing Documents and Images*, vol. 2, 1999.

[5] O. Benjelloun, H. Garcia-Molina, H. Gong, H. Kawai, T. E. Larson, D. Menestrina, and S. Thavisomboon, "D-swoosh: A family of algorithms for generic, distributed entity resolution," in *Distributed Computing Systems, 2007. ICDCS'07. 27th International Conference on*. IEEE, 2007, pp. 37–37.

[6] S. P. Benny, S. Vasavi, and P. Anupriya, "Hadoop framework for entity resolution within high velocity streams," *Procedia Computer Science*, vol. 85, pp. 550–557, 2016.

[7] C. Böhm, G. de Melo, F. Naumann, and G. Weikum, "Linda: distributed web-of-data-scale entity matching," in *Proceedings of the 21st ACM international conference on Information and knowledge management*. ACM, 2012, pp. 2104–2108.

[8] D. G. Brizan and A. U. Tansel, "A. survey of entity resolution and record linkage methodologies," *Communications of the IIMA*, vol. 6, no. 3, p. 5, 2006.

[9] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache flink: Stream and batch processing in a single engine," *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, vol. 36, no. 4, 2015.

[10] D. Chen, C. Shen, J. Feng, and J. Le, "An efficient parallel top-k similarity join for massive multidimensional data using spark," *International Journal of Database Theory and Application*, vol. 8, no. 3, pp. 57–68, 2015.

[11] G. Chen, K. Yang, L. Chen, Y. Gao, B. Zheng, and C. Chen, "Metric similarity joins using mapreduce," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 3, pp. 656–669, 2017.

[12] X. Chen, "Crowdsourcing entity resolution: a short overview and open issues." in *GvD*, 2015, pp. 72–77.

[13] A. Ching, "Giraph: Production-grade graph processing infrastructure for trillion edge graphs," *ATPESC*, vol. 14, 2014.

[14] P. Christen, "Febrl – an open source data cleaning, deduplication and record linkage system with a graphical user interface," in *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2008, pp. 1065–1068.

[15] P. Christen, *Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection*. Springer Science & Business Media, 2012.

[16] P. Christen, "A survey of indexing techniques for scalable record linkage and deduplication," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 24, no. 9, pp. 1537–1555, 2012.

[17] X. Chu, I. F. Ilyas, and P. Koutris, "Distributed data deduplication," *Proceedings of the VLDB Endowment*, vol. 9, no. 11, pp. 864–875, 2016.

[18] G. Dal Bianco, R. Galante, and C. A. Heuser, "A fast approach for parallel deduplication on multicore processors," in *Proceedings of the 2011 ACM Symposium on Applied Computing*. ACM, 2011, pp. 1027–1032.

[19] A. Das Sarma, Y. He, and S. Chaudhuri, "Clusterjoin: a similarity joins framework using map-reduce," *Proceedings of the VLDB Endowment*, vol. 7, no. 12, pp. 1059–1070, 2014.

[20] D. Deng, G. Li, S. Hao, J. Wang, and J. Feng, "Massjoin: A mapreduce-based method for scalable string similarity joins," in *30th International Conference on Data Engineering (ICDE)*. IEEE, 2014, pp. 340–351.

[21] M.-M. Deza and E. Deza, *Dictionary of distances*. Elsevier, 2006.

[22] L. R. Dice, "Measures of the amount of ecologic association between species," *Ecology*, vol. 26, no. 3, pp. 297–302, 1945.

[23] M. Dillon, *Introduction to modern information retrieval*. Pergamon, 1983.

[24] X. Dong, A. Halevy, and J. Madhavan, "Reference reconciliation in complex information spaces," in *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, 2005, pp. 85–96.

[25] C. Dou, Y. Cui, D. Sun, R. Wong, M. Atif, G. Li, and R. Ranjan, "Unsupervised blocking and probabilistic parallelisation for record matching of distributed big data," *The Journal of Supercomputing*, pp. 1–23, 2017.

[26] U. Draisbach, F. Naumann, S. Szott, and O. Wonneberg, "Adaptive windows for duplicate detection," in *28th International Conference on Data Engineering (ICDE)*. IEEE, 2012, pp. 1073–1083.

[27] V. Efthymiou, G. Papadakis, G. Papastefanatos, K. Stefanidis, and T. Palpanas, "Parallel meta-blocking for scaling entity resolution over big heterogeneous data," *Information Systems*, vol. 65, pp. 137–157, 2017.

[28] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, "Duplicate record detection: A survey," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 19, no. 1, pp. 1–16, 2007.

[29] T. Elsayed, J. Lin, and D. W. Oard, "Pairwise document similarity in large collections with mapreduce," in *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers*. Association for Computational Linguistics, 2008, pp. 265–268.

[30] I. P. Fellegi and A. B. Sunter, "A theory for record linkage," *Journal of the American Statistical Association*, vol. 64, no. 328, pp. 1183–1210, 1969.

[31] E. Friedman and K. Tzoumas, *Introduction to Apache Flink: Stream Processing for Real Time and Beyond*. O'Reilly Media, Inc., 2016.

[32] S. Fries, B. Boden, G. Stepien, and T. Seidl, "Phidj: Parallel similarity self-join for high-dimensional vector data with mapreduce," in *30th International Conference on Data Engineering (ICDE)*. IEEE, 2014, pp. 796–807.

[33] A. Gal, "Uncertain entity resolution: re-evaluating entity resolution in the big data era: tutorial," *Proceedings of the VLDB Endowment*, vol. 7, no. 13, pp. 1711–1712, 2014.

[34] L. Getoor and A. Machanavajjhala, "Entity resolution: theory, practice & open challenges," *Proceedings of the VLDB Endowment*, vol. 5, no. 12, pp. 2018–2019, 2012.

[35] L. Getoor and A. Machanavajjhala, "Entity resolution for big data," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2013, pp. 1527–1527.

[36] D. Gomes Mestre and C. E. S. Pires, "Improving load balancing for mapreduce-based entity matching," in *IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2013, pp. 618–624.

[37] L. Gu, R. Baxter, D. Vickers, and C. Rainsford, "Record linkage: Current practice and future directions," *CSIRO Mathematical and Information Sciences Technical Report*, vol. 3, p. 83, 2003.

[38] M. A. Hernández and S. J. Stolfo, "The merge/purge problem for large databases," in *ACM Sigmod Record*, vol. 24, no. 2. ACM, 1995, pp. 127–138.

[39] M. A. Hernández and S. J. Stolfo, "Real-world data is dirty: Data cleansing and the merge/purge problem," *Data mining and knowledge discovery*, vol. 2, no. 1, pp. 9–37, 1998.

[40] M. Herschel, F. Naumann, S. Szott, and M. Taubert, "Scalable iterative graph duplicate detection," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 24, no. 11, pp. 2094–2108, 2012.

[41] S.-C. Hsueh, M.-Y. Lin, and Y.-C. Chiu, "A load-balanced mapreduce algorithm for blocking-based entity-resolution with multiple keys," in *Proceedings of the Twelfth Australasian Symposium on Parallel and Distributed Computing*, vol. 152. Australian Computer Society, Inc., 2014, pp. 3–9.

[42] T. Huang and S. Russell, "Object identification: A bayesian analysis with application to traffic surveillance," *Artificial Intelligence*, vol. 103, no. 1-2, pp. 77–93, 1998.

[43] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," in *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. ACM, 1998, pp. 604–613.

[44] P. Jaccard, "Étude comparative de la distribution florale dans une portion des alpes et des jura," *Bull Soc Vaudoise Sci Nat*, vol. 37, pp. 547–579, 1901.

[45] E. H. Jacox and H. Samet, "Metric space similarity joins," *ACM Transactions on Database Systems (TODS)*, vol. 33, no. 2, p. 7, 2008.

[46] Y. Jiang, D. Deng, J. Wang, G. Li, and J. Feng, "Efficient parallel partition-based algorithms for similarity search and join with edit distance constraints," in *Proceedings of the Joint EDBT/ICDT 2013 Workshops*. ACM, 2013, pp. 341–348.

[47] L. Jin, C. Li, and S. Mehrotra, "Efficient record linkage in large data sets," in *Eighth International Conference on Database Systems for Advanced Applications*. IEEE, 2003, pp. 137–146.

[48] H. Kardes, D. Konidena, S. Agrawal, M. Huff, and A. Sun, "Graph-based approaches for organization entity resolution in mapreduce," *Graph-Based Methods for Natural Language Processing*, p. 70, 2013.

[49] H. Kawai, H. Garcia-Molina, O. Benjelloun, D. Menestrina, E. Whang, and H. Gong, *P-swoosh: Parallel algorithm for generic entity resolution*. Stanford, 2006.

[50] H.-s. Kim and D. Lee, "Parallel linkage," in *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*. ACM, 2007, pp. 283–292.

[51] T. Kirsten, L. Kolb, M. Hartung, A. Groß, H. Köpcke, and E. Rahm, "Data partitioning for parallel entity matching," *Proceedings of the VLDB Endowment*, 2010.

[52] L. Kolb, H. Köpcke, A. Thor, and E. Rahm, "Learning-based entity resolution with mapreduce," in *Proceedings of the third international workshop on Cloud data management*. ACM, 2011, pp. 1–6.

[53] L. Kolb and E. Rahm, "Parallel entity resolution with dedoop," *Datenbank-Spektrum*, vol. 13, no. 1, pp. 23–32, 2013.

[54] L. Kolb, A. Thor, and E. Rahm, "Block-based load balancing for entity resolution with mapreduce," in *Proceedings of the 20th ACM international conference on Information and knowledge management*. ACM, 2011, pp. 2397–2400.

[55] L. Kolb, A. Thor, and E. Rahm, "Dedoop: efficient deduplication with hadoop," *Proceedings of the VLDB Endowment*, vol. 5, no. 12, pp. 1878–1881, 2012.

[56] L. Kolb, A. Thor, and E. Rahm, "Load balancing for mapreduce-based entity resolution," in *28th International Conference on Data Engineering (ICDE)*. IEEE, 2012, pp. 618–629.

[57] L. Kolb, A. Thor, and E. Rahm, "Multi-pass sorted neighborhood blocking with mapreduce," *Computer Science-Research and Development*, vol. 27, no. 1, pp. 45–63, 2012.

[58] L. Kolb, A. Thor, and E. Rahm, "Don't match twice: redundancy-free similarity computation with mapreduce," in *Proceedings of the Second Workshop on Data Analytics in the Cloud*. ACM, 2013, pp. 1–5.

[59] H. Köpcke and E. Rahm, "Frameworks for entity matching: A comparison," *Data & Knowledge Engineering*, vol. 69, no. 2, pp. 197–210, 2010.

[60] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, vol. 10, no. 8, 1966, pp. 707–710.

[61] W. Liu, Y. Shen, and P. Wang, "An efficient mapreduce algorithm for similarity join in metric spaces," *The Journal of Supercomputing*, vol. 72, no. 3, pp. 1179–1200, 2016.

[62] K. Ma, F. Dong, and B. Yang, "Large-scale schema-free data deduplication approach with adaptive sliding window using mapreduce," *The Computer Journal*, vol. 58, no. 11, pp. 3187–3201, 2015.

[63] K. Ma and B. Yang, "Parallel nosql entity resolution approach with mapreduce," in *International Conference on Intelligent Networking and Collaborative Systems*. IEEE, 2015, pp. 384–389.

[64] J. Makhoul, F. Kubala, R. Schwartz, R. Weischedel *et al.*, "Performance measures for information extraction," in *Proceedings of DARPA broadcast news workshop*, 1999, pp. 249–252.

[65] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 2010, pp. 135–146.

[66] P. Malhotra, P. Agarwal, and G. Shroff, "Graph-parallel entity resolution using lsh & imm." in *EDBT/ICDT Workshops*, 2014, pp. 41–49.

[67] A. McCallum, K. Nigam, and L. H. Ungar, "Efficient clustering of high-dimensional data sets with application to reference matching,"

in *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2000, pp. 169–178.

[68] N. McNeill, H. Kardes, and A. Borthwick, "Dynamic record blocking: efficient linking of massive databases in mapreduce," in *Proceedings of the 10th International Workshop on Quality in Databases (QDB)*, 2012.

[69] D. G. Mestre, C. E. Pires, and D. C. Nascimento, "Adaptive sorted neighborhood blocking for entity matching with mapreduce," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. ACM, 2015, pp. 981–987.

[70] D. G. Mestre, C. E. S. Pires, D. C. Nascimento, A. R. M. de Queiroz, V. B. Santos, and T. B. Araujoa, "An efficient spark-based adaptive windowing for entity matching," *The Journal of Systems and Software*, vol. 128, pp. 1–10, 2017.

[71] A. Metwally and C. Faloutsos, "V-smart-join: A scalable mapreduce framework for all-pair similarity joins of multisets and vectors," *Proceedings of the VLDB Endowment*, vol. 5, no. 8, pp. 704–715, 2012.

[72] G. Navarro, E. Sutinen, J. Tanninen, and J. Tarhio, "Indexing text with approximate q-grams," in *Annual Symposium on Combinatorial Pattern Matching*. Springer, 2000, pp. 350–363.

[73] J. S. Olsson and D. W. Oard, "Improving text classification for oral history archives with temporal domain knowledge," in *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2007, pp. 623–630.

[74] G. Papadakis, J. Svirsky, A. Gal, and T. Palpanas, "Comparative analysis of approximate blocking techniques for entity resolution," *Proceedings of the VLDB Endowment*, vol. 9, no. 9, pp. 684–695, 2016.

[75] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker, "A comparison of approaches to large-scale data analysis," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. ACM, 2009, pp. 165–178.

[76] R. Pita, C. Pinto, P. Melo, M. Silva, M. Barreto, and D. Rasella, "A spark-based workflow for probabilistic record linkage of healthcare data," *EDBT Workshops*, 2017.

[77] C. Rong, W. Lu, X. Du, and X. Zhang, "Efficient duplicate detection on cloud using a new signature scheme," in *Web-Age Information Management*. Springer, 2011, pp. 251–263.

[78] S. Salihoglu and J. Widom, "Gps: A graph processing system," in *Proceedings of the 25th International Conference on Scientific and Statistical Database Management*. ACM, 2013, p. 22.

[79] W. Santos, T. Teixeira, C. Machado, W. Meira, A. S. Da Silva, D. Ferreira, and D. Guedes, "A scalable parallel deduplication algorithm," in *19th International Symposium on Computer Architecture and High Performance Computing*. IEEE, 2007, pp. 79–86.

[80] S. Sarawagi, "Special issue on data cleaning," *IEEE Data Engineering Bulletin*, vol. 23, no. 4, pp. 2–3, 2000.

[81] S. Sarawagi and A. Bhamidipaty, "Interactive deduplication using active learning," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2002, pp. 269–278.

[82] T. Seidl, S. Fries, and B. Boden, "Mr-dsj: Distance-based self-join for large-scale vector data analysis with mapreduce." in *BTW*, vol. 214, 2013, pp. 37–56.

[83] Y. N. Silva and J. M. Reed, "Exploiting mapreduce-based similarity joins," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. ACM, 2012, pp. 693–696.

[84] Y. N. Silva, J. M. Reed, and L. M. Tsosie, "Mapreduce-based similarity join for metric spaces," in *Proceedings of the 1st International Workshop on Cloud Intelligence*. ACM, 2012, p. 3.

[85] K. Stefanidis, V. Efthymiou, M. Herschel, and V. Christophides, "Entity resolution in the web of data." in *WWW (Companion Volume)*, 2014, pp. 203–204.

[86] M. Stonebraker, D. Abadi, D. J. DeWitt, S. Madden, E. Paulson, A. Pavlo, and A. Rasin, "Mapreduce and parallel dbmss: friends or foes?" *Communications of the ACM*, vol. 53, no. 1, pp. 64–71, 2010.

[87] R. Vernica, M. J. Carey, and C. Li, "Efficient parallel set-similarity joins using mapreduce," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 2010, pp. 495–506.

[88] C. Wang, J. Wang, X. Lin, W. Wang, H. Wang, H. Li, W. Tian, J. Xu, and R. Li, "Mapdupreducer:

detecting near duplicates over massive datasets," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 2010, pp. 1119–1122.

[89] S. E. Whang, D. Menestrina, G. Koutrika, M. Theobald, and H. Garcia-Molina, "Entity resolution with iterative blocking," in *Proceedings of the ACM SIGMOD International Conference on Management of data*. ACM, 2009, pp. 219–232.

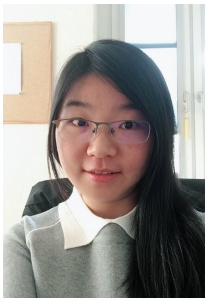
[90] W. E. Winkler, "Overview of record linkage and current research directions," in *Bureau of the Census*. Citeseer, 2006.

[91] C. Xiao, W. Wang, X. Lin, J. X. Yu, and G. Wang, "Efficient similarity joins for near-duplicate detection," *ACM Transactions on Database Systems (TODS)*, vol. 36, no. 3, p. 15, 2011.

[92] W. Yan, Y. Xue, and B. Malin, "Scalable load balancing for mapreduce-based record linkage," in *32nd International Conference on Performance Computing and Communications*. IEEE, 2013, pp. 1–10.

[93] B. Yang, H. J. Kim, J. Shim, D. Lee, and S.-g. Lee, "Fast and scalable vector similarity joins with mapreduce," *Journal of Intelligent Information Systems*, vol. 46, no. 3, pp. 473–497, 2016.

AUTHOR BIOGRAPHIES



Xiao Chen received her Msc degree in computer science from the University of Magdeburg, Germany in 2014. Currently she is a PhD student of Workgroup "Databases and Software Engineering" at the University of Magdeburg. Her research interests mainly focus on the entity resolution area, exploring open-sourced large-scale data processing frameworks or computation engines (eg. Apache Spark, Hadoop MapReduce, Apache Hive) to best support solving parallel entity resolution problems.

Eike Schallehn is a scientific assistant at the database research group of the University of Magdeburg, Germany. Since 1999 he has been doing research in the field of information integration focusing on query processing in heterogeneous and parallel environments. Other research and application fields of interest include self-tuning for databases and Cloud-based data management.

Gunter Saake is a full professor of Computer Science. He is the head of the Databases and Software Engineering Group at the University of Magdeburg, Germany. His research interests include database integration, tailor-made data management, database management on new hardware, and feature-oriented software product lines.