Noname manuscript No.
(will be inserted by the editor)

Parallel Entity Resolution with Dedoop

Preprint, accepted for publication in "Datenbank-Spektrum"

Lars Kolb · Erhard Rahm

Received: date / Accepted: date

Abstract We provide an overview of Dedoop (Deduplication with Hadoop), a new tool for parallel entity resolution (ER) on cloud infrastructures. Dedoop supports a browser-based specification of complex ER strategies and provides a large library of blocking and matching approaches. To simplify the configuration of ER strategies with several similarity metrics, training-based machine learning approaches can be employed with Dedoop. Specified ER strategies are automatically translated into MapReduce jobs for parallel execution on different Hadoop clusters. For improved performance, Dedoop supports redundancy-free multi-pass blocking as well as advanced load balancing approaches. To illustrate the usefulness of Dedoop, we present the results of a comparative evaluation of different ER strategies on a challenging real-world dataset.

Keywords MapReduce · Hadoop · Entity Resolution · Blocking · Data Skew · Load Balancing

1 Introduction

Deduplication or entity resolution (ER) is the task of identifying entities referring to the same real-world object [6]. It is a pervasive problem and of critical importance for data quality and data integration, e.g., to identify duplicate customers in enterprise databases or to match product offers for price comparison portals. ER techniques usually compare pairs of entities by evaluating multiple similarity measures to make effective match decisions. As a consequence, ER is an expensive process that can take several hours or even days [16] for large datasets as it is typical for "Big Data" applications. A common approach to improve efficiency is to reduce the search space by adopting so-called blocking techniques [2]. For example, Standard Blocking (SB) utilizes a blocking key, derived from the values of one or several entity attributes, to partition the input data into multiple candidate sets (called blocks) and restricts the subsequent matching to entities of the same block. However, ER remains a costly process and, thus, is an ideal problem to be solved in parallel on cloud infrastructures.

We present Dedoop (Deduplication with Hadoop), an ER framework based on MapReduce (MR). The MR programming model is well suited for ER because it supports the parallel matching of entities. As illustrated in Figure 1, a single MR job can be utilized for blocking-based entity resolution. Several map tasks on different nodes read the partitioned input data and apply the map function to determine the blocking key (product type in the example) per entity. All entities are dynamically redistributed among the reduce tasks such that all entities with the same key are assigned to the same reduce task. The reduce tasks perform block-wise entity resolution in parallel by comparing all entities per block with each other to determine the matching entity pairs.

While the sketched use of MR for parallel entity resolution is conceptually simple, it turns out that the manual specification of MR jobs and their deployment on different infrastructures is a tedious and time-consuming process. The parallelization of ER strategies is further complicated by the fact that different blocking approaches imply different approaches for data redistribution within MR jobs. Furthermore, MR can not always guarantee high performance, e.g., in the case of skewed block sizes (leading to load imbalances

Lars Kolb · Erhard Rahm

Institut für Informatik
Universität Leipzig
PF 100920
04009 Leipzig
Germany

E-mail: {kolb,rahm}@informatik.uni-leipzig.de

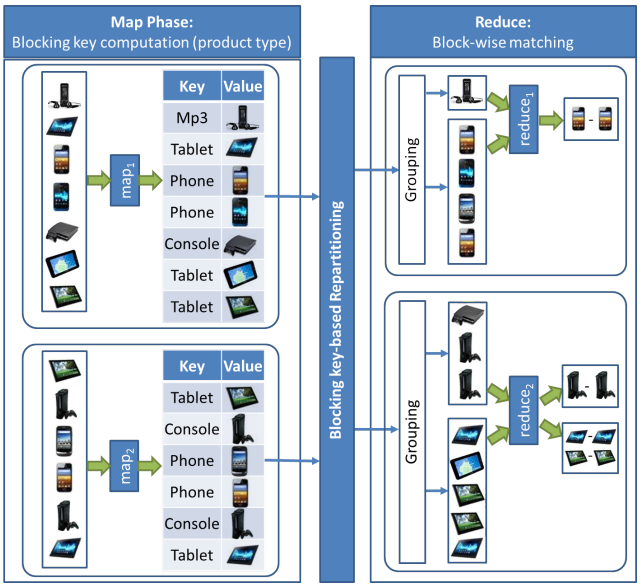


Fig. 1 Simplified MR workflow for entity resolution (product offers) using two map and two reduce tasks on different nodes.

between reduce tasks) or due to redundant pair comparisons in the case of overlapping blocks. Dedoop addresses these issues by providing the following features:

- Dedoop lets users easily specify advanced ER strategies in a Web browser. Users can thereby choose from a rich set of common ER components (e.g., blocking techniques, similarity functions etc.) including machine learning for automatically building match classifiers.
- Dedoop automatically transforms the specification into an executable MapReduce workflow and manages its submission and execution.
- Dedoop is designed to serve multiple users that may simultaneously execute multiple workflows on the same or different Hadoop clusters.
- Dedoop provides load balancing strategies to evenly utilize all nodes of the cluster. It is also able to avoid unnecessary entity pair comparisons that result from the utilization of multiple blocking keys.

Previously, we have already described several of the MR-based approaches incorporated into Dedoop, in particular the MR-based Sorted Neighborhood blocking approach [14], the load balancing approaches [13], the support of learning-based matching [10], and the approaches for avoiding redundant entity comparisons [12]. Dedoop makes these approaches usable in a unified easy-to-use system that includes additional blocking and matching approaches. In addition to providing an overview of the main features of Dedoop, we show the versatility and utility of the tool by presenting a comparative evaluation of different ER strategies on a challenging real-world dataset.

In the following, we first discuss related work and then give an overview of Dedoop (Section 3). We summarize Dedoop's techniques for load balancing in Section 4 and for avoiding redundant pair comparisons in Section 5. In Section 6, we present the evaluation results for the parallel processing of different ER strategies and analyze the scalability of Dedoop.

2 Related work

MapReduce (MR), is a programming model designed for parallelizing data-intensive computing in cluster environments [5]. MR implementations like Hadoop rely on a distributed file system (DFS) that can be accessed by all nodes. Data is represented by key-value pairs and a computation is expressed employing two user-defined functions, map and reduce, which are processed by a fixed number of map and reduce tasks.

$$map : (key_{in}, value_{in}) \rightarrow list(key_{tmp}, value_{tmp})$$
$$reduce : (key_{tmp}, list(value_{tmp})) \rightarrow list(key_{out}, value_{out})$$

For each intermediate key-value pair produced in the map phase, a target reduce task is determined by applying a partitioning function that operates on the pair's key. The reduce tasks first sort incoming pairs by their intermediate keys. The sorted pairs are then grouped and the reduce function is invoked on all adjacent pairs of the same group. This simple processing model supports an automatic parallel processing on partitioned data for many resource-intensive tasks including entity resolution.

Entity resolution is an active research topic and many approaches and frameworks have been developed and evaluated as described in recent surveys [6,15,3]. Current frameworks mostly support several methods for blocking, for matching, and for the combination of individual match results. The combination of match results may have to be manually specified or can be determined by a classification model determined by a training-based classifier such as decision tree or SVM [1]. In Dedoop, we build on this previous work but the methods in our library have been enabled for parallel processing within the MapReduce framework. We are not aware of any other system that comprehensively supports parallel entity resolution using MapReduce.

There are relatively few approaches that consider parallel entity resolution. The authors of [4] show how the match computation can be parallelized among several cores on a single node. Parallel evaluation of the Cartesian product of two sources is described in [8]. In our previous study [9], we proposed a general model for parallel entity matching based on a balanced partitioning of the input data to create match tasks that can be evaluated in parallel. This work did not consider the specifics of MR and focused on parallel matching while blocking was not performed in parallel. In addition to our own approaches utilized in Dedoop, there are a few further proposals to employ MR for ER (e.g., [24,25]).

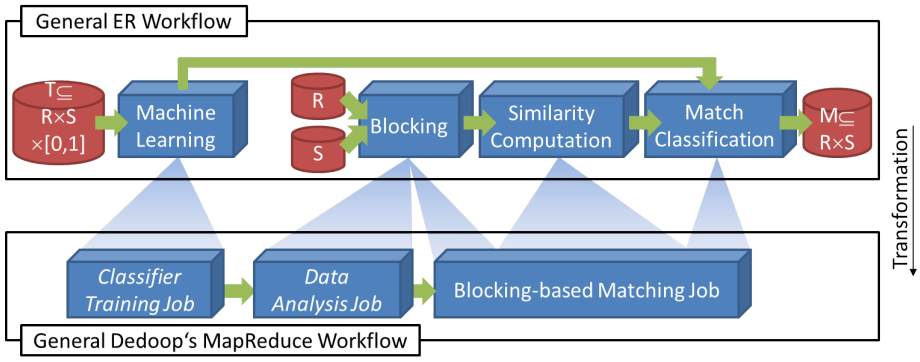


Fig. 2 Dedoop's general entity resolution workflow (upper part) and its transformation into an executable MapReduce workflow (lower part)

These approaches do not support advanced features such as load balancing or redundancy-free multi-pass blocking. They also do not support learning-based ER.

Load balancing and skew handling are well-known problems for parallel data processing but have only recently gained attention for MapReduce [21,18,19,7]. [21] presents a theoretical analysis of skew effects for MR but focuses on linear processing of entities in the reduce phase while ER has quadratic complexity to compare entities with each other. [18] proposes a load balancing scheme for scientific tasks but only deals with computational skew but not with data skew. SkewTune [19] is a generic load balancing approach that is invoked for a MapReduce job as soon as the first map (reduce) process becomes idle and no more map (reduce) tasks are pending. Then, the remaining keys (keygroups) of running tasks are tried to redistribute so that the capacity of the idle nodes is utilized. The approach in [7] is similar to our previous load balancing work [13] as it also relies on cardinality estimates determined during the map phase of the computation. This study as well as SkewTune are not focusing on entity resolution and cannot handle skew problems introduced by dominating blocks or key groups that need to be distributed among several reduce tasks.

Dedoop also supports multi-pass blocking techniques where entities can be clustered according to multiple blocking keys. On the one hand, compared to single-pass blocking, such approaches reduce the risk that matching pairs are erroneously eliminated by the pruning of the search space. On the other hand, multi-pass blocking typically results in overlapping blocks and, thus, redundant comparisons of the same entities. One of the few studies to eliminate such redundant pair comparisons is [23]. They focus on the non-distributed case and propose a complex block restructuring to avoid redundant comparisons. Another recent work for MR-based blocking and matching uses two additional MR jobs for removing duplicate entity pairs from the match result that originate from overlapping blocks [22]. Dedoop includes a method for avoiding redundant match comparisons without such additional MR jobs (Section 5).

3 Dedoop overview

The Dedoop tool [11] provides a Web interface to specify entity resolution strategies for match tasks and to schedule them on Hadoop clusters. The main steps involved for this are as follows:

- First, the files containing the input data to be matched are specified. Duplicates can be identified either within a single dataset or between two entity sets.
- The user can choose to evaluate the Cartesian product or select a blocking strategy to reduce the search space. Currently, Dedoop provides MR-adapted versions of Standard Blocking (SB) as well as of Sorted Neighborhood (SN) based on user-specified keys. Both blocking strategies allow to choose a multi-pass variation (multiple keys) for improved match quality.
- Users can select multiple similarity metrics to evaluate the similarity of entity pairs. To derive a match or non-match decision per pair, users can choose among several match classification strategies. A large set of numerical and string similarity measures, that can be applied to comparable attribute values, is supported. The match decision can be based on manually specified match rules, for example, in the form of a conjunction of similarity conditions (e.g., the manufacturer and product name of matching product offers should exceed certain thresholds) or by requesting that the weighted average of attribute-wise similarity values should exceed a threshold. Alternatively, the match decision can be determined by a classifier that has been trained by a machine learning algorithm (e.g., SVM, decision tree, or logistic regression) for which we rely on existing libraries such as *WEKA*. In the latter case, the training data has to be provided as well.
- The user can finally submit Dedoop-generated MR workflows implementing the specified entity resolution strategy on different Hadoop clusters and monitor their execution.

In the following, we provide more details on the generated MapReduce workflows and their execution.

3.1 MapReduce jobs for entity resolution with Dedoop

Figure 2 shows the general ER execution pattern of Dedoop in the upper layer for two input datasets R and S. The first step on the left is only relevant for the learning-based match approaches; it uses a subset of the entity pairs for training to learn a classification model. The main part of the ER workflow consists of three consecutive steps: blocking, similarity computation, and the actual match decision that provides the output M consisting of all pairs from the Cartesian product of input entities that are considered to match.

The lower layer of Figure 2 shows the MapReduce jobs that are generated and scheduled for execution by Dedoop for the specified entity resolution strategy. The first two jobs are optional while the last MR job, similarly as sketched in the introduction, implements parallel blocking and matching. This job is mandatory and by far the most time-consuming job. The first MR job (Classifier Training) is scheduled for learning-based matching to train a classifier based on a set of labeled examples. Dedoop employs the specified classifier and ships the resulting classification model to all nodes using Hadoop's distributed cache mechanism. The second MR job (Data Analysis) supports the load balancing algorithms of Dedoop by analyzing the data distribution for a specified blocking key. As we will see in Section 4, this analysis information helps to define a tailored data redistribution to avoid data skew effects during matching.

The details of the main MR job depend on the chosen blocking strategy, the load balancing scheme (Section 4), the selected approach for matching, and match classification. We already discussed in the introduction the use of SB that is realized within the map phase. The map function can be used to determine for every input entity its blocking key and to output a key-value pair (blocking_key, entity). The partitioning operates on the blocking keys and distributes key-value pairs among reduce tasks so that all entities sharing the same blocking key are assigned to the same reduce task. Finally, the reduce function is called for each block and computes the similarities for all entity pairs within its block.

SN is conceptually different from SB and requires the input entities to be sorted according to a selected blocking key. The assumption is that all matching entities are in close neighborhood according to the sort order of the selected key. It, thus, only compares entities within a window of a predetermined size w. A MR-based implementation of SN must ensure that reduce tasks can evaluate the entities in sort order and that the windows of neighboring entities are available despite the need to distribute entities among different reduce tasks. Dedoop uses the RepSN algorithm of [14] for this purpose. The map function determines the blocking key for each input entity and applies a specific range partitioning function to redistribute entities among reduce tasks so that the sort order according to the blocking key is preserved. The reduce tasks implement a sliding window approach for comparing the entities within their data partition. To allow for comparing entities within the window distance that spread over different reduce tasks, we extend the original map function. Entities close to partition boundaries are replicated and repartitioned not only to the respective reduce task but also to its successor (or even to further succeeding reduce tasks, if necessary for large window sizes exceeding the number of entities per reduce task).

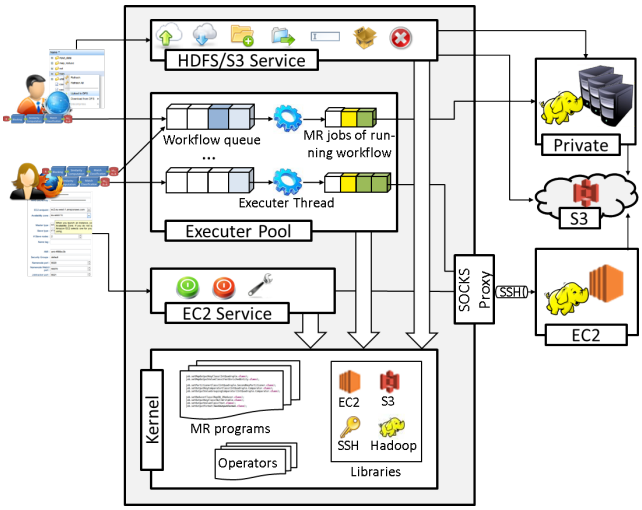


Fig. 3 Dedoop components for executing MapReduce workflows on different Hadoop clusters.

3.2 Submission and execution of MapReduce jobs

Dedoop allows multiple users to simultaneously specify and execute multiple MapReduce workflows on the same or different Hadoop clusters. Figure 3 illustrates the Dedoop components involved in the execution of MR workflows. Users can interactively submit and monitor the execution of their workflows within a browser-based GUI. Different Hadoop clusters, e.g., on local machines or Amazon EC2, can be interactively configured and launched for use with Dedoop. For each connected cluster, Dedoop maintains a workflow executer and a queue for workflows to be executed. User clients periodically poll the workflow executer pool to monitor the progress of their workflows and update the GUI with the current information about the execution. The submission of workflows and data handling is greatly simplified by user services such as an integrated graphical file manager supporting common file operations (e.g., upload, download, decompress, inspect) on HDFS and Amazon S3.

A major simplification for users is that Dedoop fully supports Hadoop clusters running on Amazon EC2. Dedoop automatically spawns and terminates SOCKS proxy servers on the machine it is hosted on to pass connections to Hadoop nodes on EC2. This is required to invoke HDFS commands

and to submit MR jobs from outside EC2, mainly due to EC2's use of internal IPs. The GUI expedites the recurring and laborious task of launching a set of virtual machines (VM) and spawning a new Hadoop cluster on it. The user therefore can specify a Linux-based Amazon Machine Image (AMI) stored in S3. The AMI must contain a distribution of Hadoop, a Java Virtual Machine, the command line utility xmlstarlet[1], and an ssh server with a secret private key. The ssh server must be configured to permit ssh access for the contained private key. On the one hand, this allows password-less ssh connections between VMs created from the AMI and on the other hand, enables Dedoop to modify the Hadoop configuration files (e.g., IP addresses, map/reduce task capacity, JVM child args) by submitting xmlstarlet commands via ssh. This approach allows for a more fine-grained configuration of the launched Hadoop cluster by Dedoop's GUI compared to using Amazon's Elastic MapReduce service.

4 Load balancing

4.1 Data skew problem

Data skew can significantly limit the scalability of MR programs. In particular, the key-based redistribution of data between map and reduce tasks can lead to large differences in the size of so-called keygroups that are to be processed by different reduce tasks. As a consequence, the scalability is limited by the time required for processing the largest keygroup. Additionally, the complexity of reduce functions, e.g., pairwise similarity computation, may be non-linear w.r.t. the input size. Hence even only slightly varying keygroup sizes can lead to noticeable runtime differences. For instance, in the initial example from Figure 1, the second reduce task has to process only three entities more than the first reduce task. However, the number of entity comparisons of the second reduce task is more than twice the number of comparisons of the first reduce task (13 vs. 6). The only load balancing mechanism supported by MapReduce is the adjustment of the partitioning function, e.g., one could assign all consoles to the first reduce task as well. Although this would balance the reduce task for the example, it does not remove the limitation that the largest keygroup marks the lower bound of the execution time. Furthermore, a detailed knowledge about data characteristics is required. Because real-world data *is skewed*, it is of crucial importance for MR-based entity resolution to distribute the processing of large blocks among several reduce tasks.

[1] `http://xmlstar.sourceforge.net/`

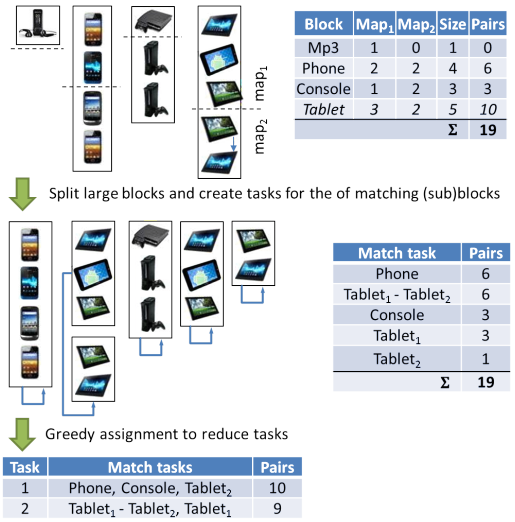


Fig. 4 BlockSplit load balancing scenario.

4.2 Dedoop's load balancing mechanism

Dedoop's load balancing approaches use an additional MR job that is executed right before the actual matching job in order to analyze the input data with respect to the specified blocking keys. The analysis job determines a so-called block distribution matrix (BDM) that indicates for every blocking key and every (map) input partition the number of entities. This information reveals block-specific data skew and allows for a unique enumeration of blocks, entities of a block, and even of entity pairs to compare. It is then used by a subsequently executed MR job that realizes load balancing in the map phase by devising a tailored data redistribution among reduce tasks that perform the entity comparisons. All of our load balancing strategies exploit the fact that entity pairs can be processed independently from each other.

Dedoop supports two load balancing schemes for SB, BlockSplit and PairRange, and one for SN within the RepSN algorithm. All approaches utilze the BDM information determined by the analysis job. We only sketch the main ideas of the approaches in the following and refer to [14, 13] for more detailed descriptions.

Figure 4 illustrates the BlockSplit load balancing approach for the example from Figure 1. The BDM, shown at the top right, specifies the number of entities and entity pairs to evaluate per block. BlockSplit focuses on large blocks that would lead to skewed execution times, i.e., those blocks whose number of pairs exceeds the average workload per reduce task. For the example and two reduce tasks, this only is the case for block (product type) *Tablet* ($10 > 19/2 = 9.5$). Such blocks are split into m sub-blocks, according to their input partitioning among the m match tasks. For split blocks, several match tasks are generated and distributed among reduce tasks. Each sub-block is (like any unsplit block) processed by a single match task. Furthermore, pairs of sub-

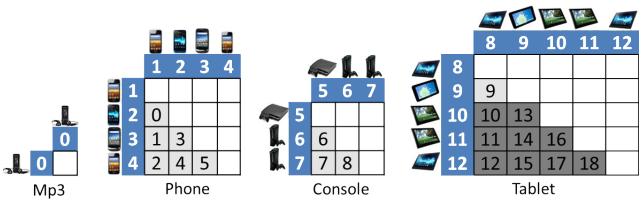


Fig. 5 PairRange's global enumeration of all pairs for the example of Figure 1. The two different shades indicate the assignment of pairs to reduce tasks.

blocks are processed by match tasks that evaluate the Cartesian product of their entities. This ensures that all comparisons of the original block will be computed in the reduce phase. For the example, we split block *Tablet* into two sub-blocks and generate three match tasks (one per sub-block and one to compare the two sub-blocks) as shown in the middle of Figure 4. The map function replicates a particular entity for each match task the entity is contained in and sends it to the responsible reduce task. BlockSplit assigns match tasks in descending order of their size among reduce tasks. This guarantees that the largest match tasks are processed first to make it unlikely that larger delays occur at the end of the computation when most nodes are already idle. For the example, we achieve a balanced distribution of the workload with 10 and 9 pairs per reduce task.

The PairRange algorithm aims at a more fine-grained control to balance the number of entity comparisons between reduce tasks. For this purpose, it virtually enumerates all entities and entity pairs to be compared based on the information from the BDM. This is illustrated in Figure 5 for our example where the 19 pairs are numbered block-wise from 0 to 18. For load balancing, PairRange splits the set of pairs into r equally-sized pair ranges and assigns the i^{th} range to reduce task i (r is the number of reduce tasks). This distribution is derived from the BDM and known to every map task so that the map function can decide to which reduce tasks an entity has to be sent or replicated. For the example, the distribution of work (pair ranges) among two reduce tasks is illustrated in Figure 5. The last matrix shows that the second tablet is needed for evaluating both pair ranges. Thus, the map function will replicate this tablet entity to the first and to the second reduce task. The evaluation in [13] showed that BlockSplit and PairRange perform similarly well. PairRange performs slightly better for large datasets and is less dependent on the initial data partitioning.

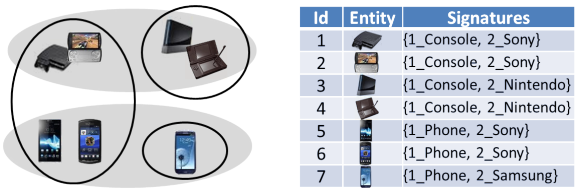
In contrast to SB, SN inherently is less susceptible to load imbalances since the number of pairs to evaluate is only determined by the window size w and independent from the blocking keys. Hence, there are no large blocks whose processing may dominate the overall execution time. However, the MapReduce realization of SN needs to ensure that the redistribution of entities among reduce tasks observes the sort order for the chosen blocking key and, thus, applies a range partitioning on the blocking keys. As a consequence, all entities having the same blocking key value will be assigned to the same reduce task. Since blocking keys of real-world data sets *are* skewed, this causes a varying number of entities per reduce task and, thus, unbalanced workloads. Even without dominating key values, it is very challenging to determine a range partitioning for data redistribution that preserves the sort order of the blocking key and leads to similarly-sized partitions per reduce task, especially for a large number of reduce tasks.

Fig. 6 Scenario for RepSN-based load balancing for Sorted Neighborhood ($r = 2$, $w = 3$).

Dedoop's RepSN approach therefore uses the information from the BDM and follows a similar entity enumeration scheme as PairRange to assign evenly-sized entity ranges to the different reduce tasks. The only difference is that the enumeration is based on the sort order of the blocking key. The map function assigns the i^{th} entity to reduce task $\lfloor i \cdot \frac{n}{r} \rfloor$ (n is the number of entities). Additionally, the last $w-1$ entities that are assigned to reduce task i have to be replicated to reduce task $i+1$ (or even to further succeeding reduce tasks if $w > \frac{n}{r}$) to allow the comparison of boundary entities that are spread over different reduce tasks. This is illustrated for our example in Figure 6 where entities are sorted according to the product type. The first six entities are assigned to the first, the remaining six to the second reduce task. For window size $w = 3$, the boundary entities 5 and 6 are replicated to the second reduce task as well to allow the comparison with entities 7 and 8.

5 Redundancy-free matching

Using a single blocking key may not sufficiently allow finding all duplicates. Especially with dirty input data and missing attribute values, matching entities can often be assigned to different blocks so that they are not compared with each other with a single-pass blocking approach. Dedoop therefore supports multi-pass blocking where entities are grouped into blocks according to multiple blocking keys derived from different attributes. The likely improvements in match quality come at the price of additional comparisons due to the increased number of blocks. However, some of these additional comparisons are redundant if the blocks overlap so that the same pairs of entities are repeatedly compared with each other. To improve the performance for multi-pass blocking, it is, thus, important to eliminate such redundant comparisons.



Id	Entity	Signatures
1		{1_Console, 2_Sony}
2		{1_Console, 2_Sony}
3		{1_Console, 2_Nintendo}
4		{1_Console, 2_Nintendo}
5		{1_Phone, 2_Sony}
6		{1_Phone, 2_Sony}
7		{1_Phone, 2_Samsung}

Fig. 7 Example of overlapping entity pairs for two-pass blocking. The entities are partitioned by the product type (gray shaded) as well as by manufacturer (black bordered).

Figure 7 illustrates the problem for a small set of seven product entities and two-pass SB. The first blocking key partitions the entities by their product type (*Console, Phone*) and the second by manufacturer (*Sony, Nintendo, Samsung*). By considering only the product type, we would miss the matching entity pair $(2,6)$ (a smartphone for gamers). However, as a consequence 3 out of 16 entity pairs overlap and they should not be compared redundantly.

Without the avoidance of redundant comparisons, the support multi-pass blocking is straight-forward. The map function can repeatedly output each entity for every pass. It thereby prefixes the blocking keys with the pass numbers to obtain a combined key[2] for data distribution among reduce tasks. Hence, all blocks of all passes are processed in parallel. To avoid redundant comparisons, we have to unambiguously determine which reduce task is "responsible" for any pair comparison. The key idea is to (virtually) enumerate the entities' candidate sets (i.e., blocks or windows) and to compare each entity pair for their smallest common candidate set index only. This can be achieved by slightly extending the map output value with some additional metadata. The enrichment of an entity with information about its candidate sets allows the reduce function to determine whether the current reduce task is responsible for comparing a certain entity pair.

For SB, this is realized as follows. The map function determines the list S of blocking keys (signatures) and outputs an (s, entity) pair for each $s \in S$. It further enriches the entity with the sublist of blocking keys that are smaller than the current key s. For two entities, the reduce function checks whether their signature sublists overlap. The evaluation of the pair is skipped when there is a common blocking key in the sublists (smaller than the current one) since this indicates that another reduce task performs the comparison. For our example in Figure 7, there will be two map outputs per product entity, one for the product type and one for the manufacturer blocking key. For the first entity, map outputs a pair (1_Console, [1, ∅]), i.e. the sublist for entity 1 is empty for key 1_Console. Additionally, map outputs a (2_Sony, [1, {1_Console}]) pair, i.e., the sublist indicates that the entity's blocking key 1_Console is (lexicographically) smaller than the current key 2_Sony. Similarly, map outputs for the second entity the pairs (1_Console, [2, ∅]) and (2_Sony, [2, {1_Console}]). With this information, reduce skips the comparison of both entities for the signature 2_Sony since they have a common smaller signature.

For SN, there is a slight modification since entity pairs are not considered based on a common blocking key but on whether their relative position in the key's sort order is within the window distance. Therefore, we annotate entities not with their (smaller) blocking keys but by their position index in the sorted entity list for each preceding pass. For an entity pair, the reduce function then checks the two lists whether for a preceding pass the absolute difference of the two indexes is less than window size w. In this case, the comparison is skipped as it is already performed for a previous pass.

The evaluation of the approaches in [12] showed that the additional checks in the reduce function are by far outweighed by the savings of avoiding redundant comparisons and match decisions.

6 Evaluation

To demonstrate the versatility and efficiency of Dedoop, we analyze the results for two experiment series. The first experiment uses Dedoop for a comparative evaluation of different blocking and matching approaches w.r.t. both match quality and runtime for the same real-world ER task on the same MapReduce platform. The second experiment focuses on efficiency and scalability for large Hadoop clusters.

Both experiments are conducted on Amazon EC2 using worker nodes of type c1.medium (providing 2 virtual cores) and a dedicated master instance of type m1.small. The utilized VMs are based on Ubuntu Server 12.04 with an Oracle Java 1.6 64-bit JVM, and Hadoop 0.20.2. Each node runs two map (reduce) processes to execute map (reduce) tasks. For load balancing reasons, we generate 5 times as many reduce tasks than processes (i.e., 10 per node) for the MR jobs computing the match comparisons. The use of ECs complicates the measurements of execution times since EC2 provides different hardware within the same instance type. We therefore generally execute the runs repeatedly using a new set of VMs per run and average the observed execution times.

6.1 Comparative evaluation of different ER strategies

In this experiment, we investigate ER strategies for the real-world dataset GS from [16] containing 64,263 publication

[2] Internally, Dedoop prefixes each blocking key with its (zero-padded) pass number to force blocking keys of pass i to be lexicographically smaller than keys of pass $j > i$. In favor of readability, this has been omitted in the previous sections.

	Standard Blocking								Sorted Neighborhood								Cartesian			
	Single pass (author)				Multi-pass (author, title)				Single pass (author/w=1000)				Multi-pass (author/w_1=500, (title/w_2=500)							
	$Rule_1$	$Rule_2$	SVM	DT	$Rule_1$	$Rule_2$	SVM	DT	$Rule_1$	$Rule_2$	SVM	DT	$Rule_1$	$Rule_2$	SVM	DT	$Rule_1$	$Rule_2$	SVM	DT
Comparisons	≈3,7 · 10^6				≈23,2 · 10^6				≈63.7 · 10^6				≈63.9 · 10^6				≈2.06 · 10^9			
Time [in min]	1.73	1.77	2.21	2.14	1.97	2.03	2.57	2.58	2.48	2.48	3.34	2.73	2.68	2.67	3.35	2.74	24.00	25.08	44.61	35.94
Recall	25.5%	45.2%	48.6%	55.2%	45.7%	57.2%	73.0%	79.6%	28.2%	50.4%	52.7%	59.3%	44.8%	62.1%	72.4%	78.9%	52.2%	69.6%	93.5%	92.6%
Precision	96.1%	88.4%	70.4%	70.4%	86.8%	89.4%	73.3%	74.5%	95.6%	87.5%	85.8%	75.3%	87.0%	88.5%	74.1%	74.4%	84.7%	87.0%	66.0%	71.2%
F-Measure	40.3%	59.8%	57.5%	61.9%	59.9%	69.7%	73.2%	77.0%	43.6%	64.0%	65.3%	66.4%	59.1%	73.0%	72.2%	76.5%	64.6%	77.4%	77.4%	80.5%

Fig. 8 Execution times and match quality of different entity resolution approaches for the GS match task.

Match Classification	Input Similarity Features	Match Criterium
$Rule_1$	3-gram(title)	sim ≥ 0.8
$Rule_2$	3-gram(title) 3-gram (authors)	sim (title)≥ 0.6 and sim (author)≥ 0.4
SVM	3-gram(title) 3-gram (authors)	SVM (WEKA LibSVM, -K 0 -C 10)
DT	3-gram(title) 3-gram (authors)	Decision Tree (WEKA LMT, default config)

Fig. 9 Considered approaches for match classification

records from the Google Scholar search engine. The publication records are relatively unclean due to the high heterogeneity of paper citations and errors for extracting the bibliographic information from PDF files. To evaluate match quality in terms of precision, recall, and F-measure, we derived a gold standard from the manually determined, perfect match result from [16] that is based on a mapping of the GS publications to corresponding DBLP publications.

We investigate the relative effectiveness of 20 ER strategies by considering five blocking strategies (single-pass and two-pass versions for Standard Blocking (SB) and Sorted Neighborhood (SN) as well as the use of no blocking, i.e., the evaluation of the Cartesian product, and the four approaches listed in Figure 9 to derive a match decision from the computed similarities. These approaches always evaluate the trigram string similarity of the publication title and of the author attribute. The similarity computation is done on normalized attribute values in lower case after the removal of punctuation and redundant whitespace characters. As representatives for non-learning ER strategies, we consider two simple, manually specified match rules defining the minimal similarity on either only the title attribute ($rule_1$) or for both title and author ($rule_2$). We also consider two learning approaches for match classification by applying either the SVM or the decision tree classifier from the WEKA library. For training, we only use 500 entity pairs labeled with their match decision. We apply the ratio approach of [17] to select the training data and ensure that both matches and non-matches constitute at least 40% of the training samples. The reported results for the learning-based approaches are averages over ten runs with different training datasets.

The table in Figure 8 shows the obtained results for the 20 ER strategies, in particular for recall, precision, and F-Measure. We also report how many entity pairs have been compared for the different blocking approaches and the overall parallel execution time on a Hadoop cluster with 20 nodes. Naturally, the best match quality is achieved without blocking when we evaluate the complete Cartesian product of more than two billion entity pairs. The best F-Measure is only about 80% indicating the hardness of the match problem (bibliographic ER tasks involving relatively clean data sources such as DBLP are typically solved with F-measure values of more than 90-95% [16]). The best match quality is achieved for the learning-based approach using decision trees followed by the SVM-based learning approach and, with comparable quality than SVM, the use of $rule_2$. The $rule_1$ match decision that matches all publications with a title similarity of at least 80% turns out as too simple and restrictive since it achieves only a recall of at most 52%. This means that about half of the matching publications have a title similarity below 80% which is again an indicator of poor data quality.

The execution time for evaluating the Cartesian product is about 25 minutes for the non-learning approaches and 36 - 45 minutes for the learning approaches. With blocking, the number of entity pairs to evaluate significantly drops (by a factor 100 - 300) and the execution time is reduced to only 1.7 - 3.3 minutes. These times include the execution of the classifier training job (if required), the analysis job to support load balancing, as well as the MR-inherent overhead for I/O and data redistribution. We use the first three characters of the author attribute as blocking key for single-pass blocking and additionally the first three characters of the title attribute for two-pass blocking. For SN, we employ a window size w=1000 for single-pass and w=500 for two-pass blocking so that nearly the same number of comparisons result in both cases. For SB, the number of comparisons for two-pass blocking is much higher than for single-pass blocking (but lower than for SN). Still, the execution time for two-pass blocking is only slightly higher than for one-pass blocking since the actual match work resulted in only a light utilization of the 20 nodes.

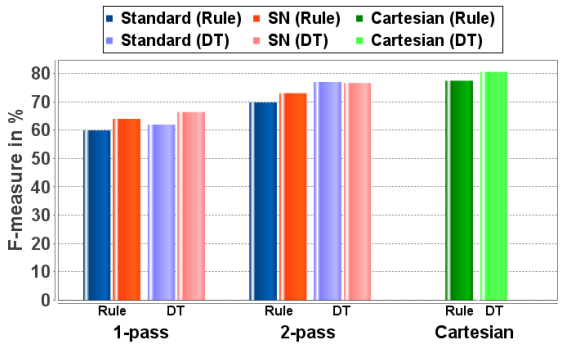


Fig. 10 Comparison of the best F-Measure results for match classification with rule_2 and decision tree (DT)

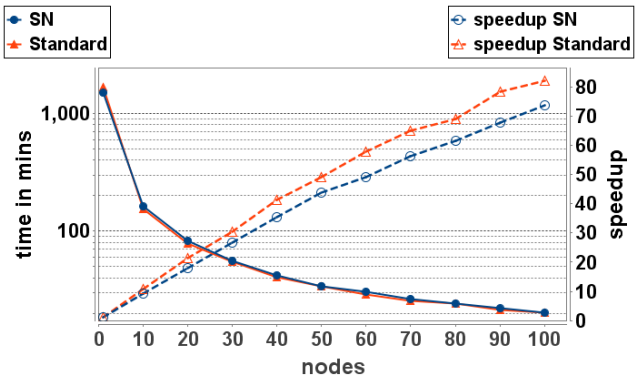


Fig. 11 Execution times and speedup using Sorted Neighborhood (SN) and Standard Blocking for the CiterseerX dataset.

The relative quality of the rule-based and learning-based ER approaches with blocking is similar to those for the Cartesian product, i.e., the decision tree classifier outperforms the other approaches. The single-pass blocking approaches based on the author attribute proved to be too restrictive and lead to a low recall and reduced F-Measure of only 66 % or less. The two-pass blocking approaches achieve significantly improved match quality for both SB and SN (up to 77% of F-Measure).

Figure 10 summarizes the achieved F-Measure results for the best non-learning approach (using rule_2) and the best learning approach (using decision tree). We observe that the decision tree consistently outperforms the rule-based classification. SN performs slightly better than SB in three of four cases. The two-pass blocking approaches substantially outperform one-pass blocking and are almost as good as the evaluation of the Cartesian Product which is less scalable to larger match problems.

The experiment shows the usefulness of Dedoop for evaluating many strategies for parallel entity resolution but is not meant to allow general conclusions regarding the relative quality of different approaches. This is because we only evaluated a single match problem and did not systematically test different match rules, different blocking keys, different training sizes etc. Still, Dedoop facilitates such extended evaluations since the execution times are significantly faster than without parallelization and since setting up different ER strategies is simple.

6.2 Scalability

The second experiment focuses on the scalability of Dedoop for a larger match problem. We therefore vary the cluster size for entity resolution from 1 up to 100 nodes. We use the CiteseerX dataset[3] from [24] containing 1,385,532 publication records. To test Dedoop's load balancing, we apply both Sorted Neighborhood (SN) in combination with the RepSN algorithm [14] as well as Standard Blocking (SB) in combination with the PairRange algorithm [13]. For both strategies, the first three characters of the publication title form an entity's blocking key (single-pass blocking). For SN, the window size was set to $w = 5000$, resulting in $\approx 6,91 \cdot 10^9$ entity comparisons. For SB, the blocking scheme leads to $\approx 6.76 \cdot 10^9$ candidate pairs, i.e., only minimally fewer than for SN. For matching, we compute the trigram similarity on the title attribute.

Figure 11 shows the resulting execution times and speedup values. Both approaches show their ability to evenly distribute the time-intensive similarity computation across reduce tasks and nodes. SN reduces the execution time from about 25 hours with one node to merely 20 minutes with 100 nodes (speedup of ≈74). The speedup behavior is even near-linear for up to 40 nodes (≈36 for $n = 40$). The somewhat reduced speedup for larger clusters is influenced by typical factors such as increased likelihood of skewed execution times per node and reduced parallelization potential for smaller tasks. The relatively large window size leads for more nodes to smaller reduce tasks and to an increasing number of boundary entities to be replicated to several reduce tasks. SB leads to a largely similar behavior underlining that the load balancing approach is also effective in this case. The execution time is almost 28 hours for one node and reduced to 20 minutes for 100 nodes. The slower execution for one node leads to a slightly better speedup (≈82) than for SN.

7 Summary and outlook

Entity resolution for "Big Data" is very time-consuming and, thus, benefits from a parallel execution in MapReduce-based cluster environments. The presented Dedoop tool realizes this kind of parallel ER for many blocking and matching strategies. For improved performance, it supports advanced

[3] We do not have the perfect match result for this dataset so we could not use it for the evaluation of match quality.

load balancing techniques to deal with data skew and avoids redundant comparisons for multi-pass blocking. The rich UI of Dedoop makes it easy to specify entity resolution workflows. The mapping of workflows to MapReduce jobs, their parameterization, and the submission to Hadoop clusters are completely transparent to the user. The presented analysis for 20 ER strategies shows the value of Dedoop for large-scale evaluations and for determining effective strategies to solve challenging match tasks. Furthermore, we could demonstrate the scalability of Dedoop and its load balancing approaches for large EC2 clusters.

In future work, we plan to integrate new features to Dedoop such as frequency-aware [20] and active learning approaches. To further improve efficiency, we are currently working on several low-level optimizations for similarity computation such as prefix, suffix, and length filtering [26]. Another option is to support the use of the individual nodes' GPUs to speed up the calculation of string similarity measures.

References

1. Bilenko, M., Mooney, R.J.: Adaptive Duplicate Detection Using Learnable String Similarity Measures. In: KDD, pp. 39–48 (2003)
2. Christen, P.: A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication. IEEE Trans. Knowl. Data Eng. **24**(9), 1537–1555 (2012)
3. Christen, P.: Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection. Data-Centric Systems and Applications. Springer (2012)
4. Christen, P., Churches, T., Hegland, M.: Febrl - A Parallel Open Source Data Linkage System. In: PAKDD, pp. 638–647 (2004)
5. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. In: OSDI, pp. 137–150 (2004)
6. Elmagarmid, A.K., Ipeirotis, P.G., Verykios, V.S.: Duplicate Record Detection: A Survey. IEEE Trans. Knowl. Data Eng. **19**(1), 1–16 (2007)
7. Gufler, B., Augsten, N., Reiser, A., Kemper, A.: Load Balancing in MapReduce Based on Scalable Cardinality Estimates. In: ICDE, pp. 522–533 (2012)
8. Kim, H., Lee, D.: Parallel Linkage. In: CIKM, pp. 283–292 (2007)
9. Kirsten, T., Kolb, L., Hartung, M., Gross, A., Köpcke, H., Rahm, E.: Data Partitioning for Parallel Entity Matching. In: QDB (2010)
10. Kolb, L., Köpcke, H., Thor, A., Rahm, E.: Learning-based Entity Resolution with MapReduce. CloudDB, pp. 1–6 (2011)
11. Kolb, L., Thor, A., Rahm, E.: Dedoop: Efficient Deduplication with Hadoop. PVLDB **5**(12), 1878–1881 (2012)
12. Kolb, L., Thor, A., Rahm, E.: Don't Match Twice: Redundancy-free Similarity Computation with MapReduce. Tech. rep. (2012). `http://dbs.uni-leipzig.de/de/publication/redfree`
13. Kolb, L., Thor, A., Rahm, E.: Load Balancing for MapReduce-based Entity Resolution. In: ICDE, pp. 618–629 (2012)
14. Kolb, L., Thor, A., Rahm, E.: Multi-pass Sorted Neighborhood blocking with MapReduce. Computer Science - R&D **27**(1), 45–63 (2012)
15. Köpcke, H., Rahm, E.: Frameworks for entity matching: A comparison. Data Knowl. Eng. **69**(2), 197–210 (2010)
16. Köpcke, H., Thor, A., Rahm, E.: Evaluation of entity resolution approaches on real-world match problems. PVLDB **3**(1), 484–493 (2010)
17. Köpcke, H., Thor, A., Rahm, E.: Learning-Based Approaches for Matching Web Data Entities. IEEE Internet Computing **14**(4), 23–31 (2010)
18. Kwon, Y., Balazinska, M., Howe, B., Rolia, J.A.: Skew-Resistant Parallel Processing of Feature-Extracting Scientific User-Defined Functions. In: SoCC, pp. 75–86 (2010)
19. Kwon, Y., Balazinska, M., Howe, B., Rolia, J.A.: SkewTune: Mitigating Skew in MapReduce Applications. In: SIGMOD Conference, pp. 25–36 (2012)
20. Lange, D., Naumann, F.: Frequency-aware similarity measures: why Arnold Schwarzenegger is always a duplicate. In: CIKM, pp. 243–248 (2011)
21. Lin, J.: The Curse of Zipf and Limits to Parallelization: A Look at the Stragglers Problem in MapReduce. In: Workshop on Large-Scale Distributed Systems for Information Retrieval (2009)
22. McNeill, N., Kardes, H., Borthwick, A.: Dynamic Record Blocking: Efficient Linking of Massive Databases in MapReduce. In: QDB (2012)
23. Papadakis, G., Ioannou, E., Niederée, C., et al.: Eliminating the Redundancy in Blocking-based Entity Resolution Methods. In: JCDL, pp. 85–94 (2011)
24. Vernica, R., Carey, M.J., Li, C.: Efficient Parallel Set-Similarity Joins Using MapReduce. In: SIGMOD Conference, pp. 495–506 (2010)
25. Wang, C., Wang, J., Lin, X., Wang, W., Wang, H., Li, H., Tian, W., Xu, J., Li, R.: MapDupReducer: Detecting Near Duplicates over Massive Datasets. In: SIGMOD Conference, pp. 1119–1122 (2010)
26. Xiao, C., Wang, W., Lin, X., Yu, J.X.: Efficient Similarity Joins for Near Duplicate Detection. In: WWW, pp. 131–140 (2008)