Don't Match Twice: Redundancy-free Similarity Computation with MapReduce

Lars Kolb
Database Group
University of Leipzig
kolb@informatik.uni-leipzig.de

Andreas Thor
Database Group
University of Leipzig
thor@informatik.uni-leipzig.de

Erhard Rahm
Database Group
University of Leipzig
rahm@informatik.uni-leipzig.de

ABSTRACT

To improve the effectiveness of pair-wise similarity computation, state-of-the-art approaches assign objects to multiple overlapping clusters. This introduces redundant pair comparisons when similar objects share more than one cluster. We propose an approach that eliminates such redundant comparisons and that can be easily integrated into existing MapReduce implementations. We evaluate the approach on a real cloud infrastructure and show its effectiveness for all degrees of redundancy.

Categories and Subject Descriptors

H.3.4 [**Systems and Software**]: Distributed Systems

General Terms

Algorithms, Performance

Keywords

MapReduce, Pair-wise similarity computation, Redundancy

1. INTRODUCTION

Pair-wise similarity computation (PSC) is an important aspect of many data-intensive applications, e.g., identifying similar documents for clustering [10], efficient set-similarity joins in databases [16], or identifying duplicates (entity resolution) [8]. PSC usually is an expensive operation because it is inherently of $O(n^2)$ complexity and typically involves complex (string) similarity functions. Therefore, it particularly benefits from the parallel MapReduce (MR) model and we observe an increasing number of MapReduce-based PSC implementations [4, 15, 3, 6, 14, 11].

The naïve approach for PSC examines the complete Cartesian product of object pairs. The resulting quadratic complexity is intolerable for large datasets even when using MR. A common approach is pruning the search space to avoid processing pairs with presumably low similarity. This is achieved by grouping objects into (possibly overlapping) clusters and restricting similarity computation to objects of the same data cluster. To this end, many MR implementations follow a similar strategy: For each object (e.g., document, string, or entity) one or more *signatures* (e.g., terms, tokens, or blocking keys) are generated. A signature identifies a particular cluster and objects are assigned to all clusters of their signatures. The map phase emits a (key=signature, value=object) pair for each signature. The MR framework then groups all pairs based on their key (signature) and thus groups together objects of the same cluster. The actual similarity (match) computation is performed within the reduce phase, i.e., all objects of the same clusters are compared with each other.

The creation of appropriate signatures for objects is difficult because it has to balance between efficiency and data quality. On the one hand, cluster sizes should be as small as possible to reduce the number of pairs and thus increase efficiency. On the other hand, small cluster sizes tend to miss similar object pairs especially for dirty (web) data. For example, if customer objects are clustered by their location, wrong or missing zip codes may place very similar objects into different clusters. Many approaches such as document clustering [10], entity resolution [8], or sequence alignment [14] therefore make use of multiple signatures per object to ensure that similar objects are still grouped together for comparison even in the presence of data quality issues.

For example, entity resolution approaches frequently apply standard blocking [2] for generating blocking keys (signatures) based on the values of one or several entity attributes. Blocking keys for finding duplicates customers in enterprise databases can be the first three letters of the customer's name or zip code. Due to common data quality issues, e.g., missing or false zip code, it is of crucial importance to utilize several blocking keys (multi-pass blocking) to achieve sufficient match pair completeness and thus match quality compared to single-pass blocking.

The sketched naïve MR-based implementation for PSC is unaware of redundancy introduced by multiple signatures per object. If an object pair shares more than one signature, it will be redundantly compared by several reduce tasks that are likely to be executed on different nodes. This unnecessary computation obviously deteriorates the run-time efficiency. Consequently, eliminating redundant pair comparison is a promising optimization approach.

DanaC'13, June 23, 2013, New York, NY, USA

We make the following contributions in this paper:

- We review how the MapReduce model can be applied to the parallel execution of clustering-based PSC. (Section 2)
- We propose an approach for the elimination of redundant pair comparisons in the common case of overlapping clusters and show how it is efficiently integrated with a parallel MR implementation. (Sections 3 and 4)
- We evaluate our approach and demonstrate its efficiency in comparison to a redundancy-unaware implementation. The experiments show that overlapping clusters increase the quality of PSC with only a moderate increase of the overall computation cost. (Section 5)

2. SIGNATURE-BASED PSC WITH MAP-REDUCE

We first formalize the problem of signature-based pairwise similarity computation. Given a set of objects O and a set of signatures S, we assume the existence of a signature function $\sigma : O \rightarrow \mathcal{P}(S)$ that assigns a (non-empty) set of signatures $s \subseteq S$ to every object $o \in O$. A PSC program then computes the similarity for all object pairs that share at least one signature, i.e., for $\{(o_1, o_2) | (o_1, o_2) \in O \times O \wedge o_1 \neq o_2 \wedge \sigma(o_1) \cap \sigma(o_2) \neq \emptyset\}$.

Figure 1 (left) shows an example for 9 objects (A-I) and 5 different signatures (1-5). The signature function could be the result of a 2-pass-blocking for entity resolution. A first pass generates three clusters or blocks (signatures 1, 2, and 3) and a second pass generates two additional blocks (4 and 5). The resulting signature function is shown in the table, e.g., object A has signatures 1 and 4. The function in our example assigns the same number of signatures to each object which is, of course, not required generally.

The basic MR implementation of signature-based PSC is as follows and Figure 1 (right) illustrates the MR dataflow for our example using two map and two reduce tasks. The map function is called for every input object and emits a (key=signature, value=object) pair for each of the object's signatures. For example, map emits two pairs (1,A) and (4,A) for object A because of $\sigma(A) = \{1, 4\}$. The partitioning distributes the key-value-pairs to the available reduce tasks. Our example employs a simple "modulo"' partitioner that sends key groups 1, 3, and 5 to the first reduce tasks and the remaining groups to the second task. The reduce function is then called for each key (=signature) and performs a pair-wise comparison for all associated objects. For example, all 6 pairs (A-B, A-C, ..., C-D) of objects sharing signature 1 are processed by the first reduce task.

Overall, 26 pairs are considered for pair-wise similarity computation. Due to the overlapping blocks, four pairs (A-B, C-D, F-G, and H-I) share two signature values and thus are considered twice. For example, pair A-B is compared twice due to the two signatures (keys) 1 and 4. The second occurrence of pair A-B (key=4) therefore is considered redundant. All redundant pairs are underlined in Figure 1. Figure 1 also illustrates that redundant pairs and their counterparts need not to be sent to the same node in the MR cluster. Thus, it requires a modification of the MR program to make reduce tasks aware of the existence of redundant pairs to avoid the unnecessary costs of repeated pair comparisons.

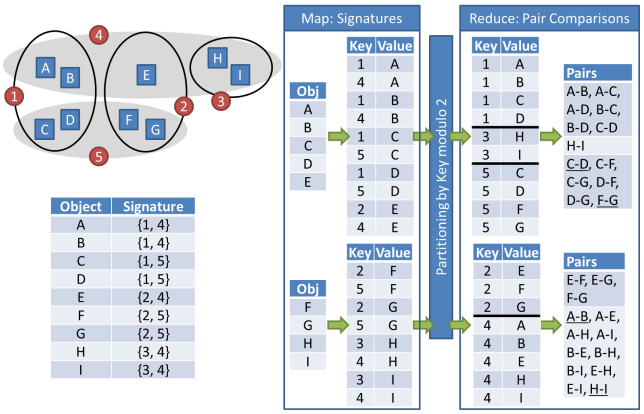

Figure 1: Example of PSC using a 2-pass blocking of 9 objects (A-I) in 3 (black bordered) and 2 (grey shaded) clusters, respectively. The resulting signature function is shown in the table. The MR computation scheme on the right side indicates that 4 (underlined) out of 26 pairs are compared redundantly.

3. MAP-REDUCE-BASED AVOIDANCE OF REDUNDANT PAIRS

Redundancy-free PSC requires that any two objects are compared only once even if they share more than one signature. The key challenge for MR-based PSC is that reduce tasks are not aware that another reduce task might process the exact same object pair. For any object pair (o_1, o_2), all reduce tasks must therefore deterministically choose a single signature $l \in \sigma(o_1) \cap \sigma(o_2)$. Our approach determines the minimal signature value of the intersection of both sets of signatures, i.e., $l = min(\sigma(o_1) \cap \sigma(o_2))$. (We will consider other deterministic functions in future work.) The pair (o_1, o_2) is then solely processed by the reduce task that handles signature l. This simple but effective approach ensures that all relevant objects pairs are considered and moreover no redundant pair comparisons are performed.

For the MR-based realization it is thus sufficient to check for any pair (o_1, o_2) of the reduce partition with key $= k$ if there is a *smaller* common signature $k' < k$ for both objects. If so, then pair (o_1, o_2) can be skipped. If not, then k is the least common signature and the current reduce task is "responsible" for the pair comparison. This can be realized by changing map and reduce as follows:

map: For every object o, map determines its signatures, i.e., $\sigma(o) = \{s_1, s_2, ..., s_n\}$. The map function does not merely emit n key-value-pairs (s_i, o) for $1 \leq i \leq n$ but it annotates the objects o with the subset of *smaller* signatures. Let $\sigma_{s_i}(o) = \{s \in \sigma(o) | s < s_i\}$, map emits key-value-pairs $(s_i, [o, \sigma_{s_i}(o)])$ for every $1 \leq i \leq n$.

reduce: The reduce phase then introduces an additional check for the current reduce key k and any pair $([o_1, \sigma_k(o_1)], [o_2, \sigma_k(o_2)])$ whether or not the two sets are disjoint, i.e., $\sigma_k(o_1) \cap \sigma_k(o_2) = \emptyset$. If they are disjoint, k is the least common signature value and the two objects o_1, o_2 are compared. Otherwise, there is a smaller common signature value $k' < k$ and the object pair (o_1, o_2) is not considered for k.

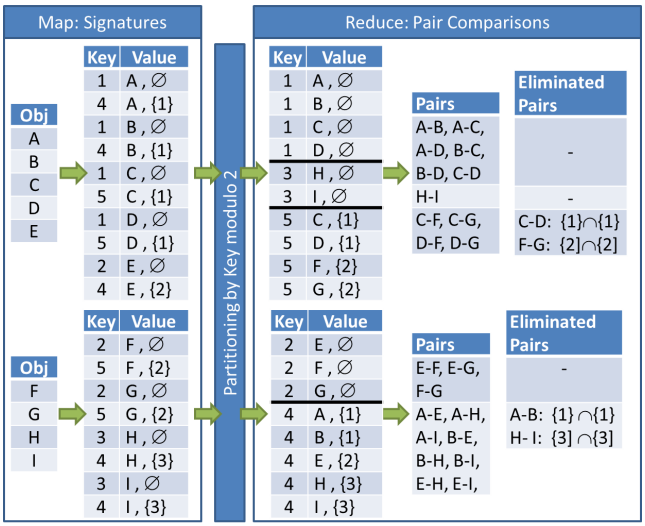

Figure 2: Example of redundancy-free MR-based PSC using the same data as in Figure 1. The result of the reduce phase illustrates both the actual compared pairs as well as the eliminated / redundant pairs.

Figure 2 illustrates the MapReduce dataflow with the avoidance of redundant pair comparisons using the same data as in Figure 1. Consider object A with signatures $\sigma(A) = \{1, 4\}$. The map function emits $(1, [A, \emptyset])$ since there is no smaller signature than 1. Additionally, map emits $(4, [A, \{1\}])$ because $\sigma_4(A) = \{1\}$. The first key-value-pair $(1, [A, \emptyset])$ is sent to the first reduce task and object A is compared to all other objects sharing key=1 (B, C, D) because there is no signature overlap. The second key-value-pair $(4, [A, \{1\}])$ is sent to the second reducer and A is supposed to be compared to B, E, H, and I since they all share key=4. A is eventually compared to E, H, and I since their signatures σ_4 ($\{2\}$ and $\{3\}$, respectively) are disjoint to $\sigma_4(A) = \{1\}$. However, the pair A-B is not considered since their sets overlap, i.e., $\sigma_4(A) \cap \sigma_4(B) \neq \emptyset$. Overall, 4 out of 26 ($\approx 15\%$) pairs are eliminated as redundant in our example.

Our approach is inspired by the *least common block index* property as used for comparison propagation in [12]. The authors focus on multi-pass blocking and similar to our approach, object pairs need only to be compared within their least common block. In contrast to our work, the authors study this problem in a non-distributed environment and they address the problem by a complex rearrangement of the initial clustering (see Section 6). Our approach does not rely on such a re-clustering but is capable of parallel MR processing and efficiently identifies redundant pairs that can be skipped at run-time.

4. IMPLEMENTATION ASPECTS

We briefly describe some implementation details of our approach. Algorithm 1 illustrates our MR implementation of redundancy-free PSC. The implementation makes use of a sorted signature list (line #3) instead of an (unsorted) set. The advantage of a sorted list is two-fold: First, the set of smaller signatures is simply the prefix list through the current signature value (lines #4ff). Second, it makes the

Algorithm 1: Redundancy-free PSC

```
1  map(k_in=unused, v_in=o)
2      S ← σ(o).distinct();
3      S.sort();
4      SS ← [] // smaller signature list
5      foreach s_i ∈ S do
6          output(k_tmp = s_i, v_tmp= (o, SS));
7          SS.append(s_i);
8  reduce(k_tmp=s, list(v_tmp)=list(object, SS))
9      buf ← {};
10     foreach (o_1, SS_1) ∈ list(object, SS) do
11         foreach (o_2, SS_2) ∈ buf do
12             if ¬doOverlap (SS_1, SS_2) then
13                 compare(o_1, o_2);
14         buf ← buf ∪ {(o_1, SS_1)};
```

Algorithm 2: Efficient overlap check

```
1  doOverlap(SS_1, SS_2)
2      i ← 0;
3      j ← 0;
4      l_1 ← SS_1.length();
5      l_2 ← SS_2.length();
6      while (i < l_1) ∧ (j < l_2) do
7          s_1 ← SS_1.get(i);
8          s_2 ← SS_2.get(j);
9          cmp ← s_1.compareTo(s_2);
10         if cmp = 0 then
11             return true;
12         else if cmp < 0 then
13             i++;
14         else
15             j++;
16     return false;
```

Figure 3: Implementation of redundancy-free MR-based PSC.

test, whether two sets overlap, more efficient (see line #12 and Algorithm 2, respectively). Since both lists are sorted the overlap check can make use of interleaved linear scans to find overlapping values.

An alternative approach to sorted signature lists is recomputing the objects' signatures in the reduce function from their attribute values. At the first glance this looks very promising because it may shrink the map output and, thus, may reduce the amount of data shuffled between the map and reduce phase. However, based on our experience this approach has several drawbacks. First, recomputing object signatures in the reduce phase requires to retain the original attribute values that in turn increases the map output. Second, the signature function needs to be called for every key-value-pair in the reduce function. The number of function calls is thus much higher than the number of objects because map emits a key-value-pair for every object signature value. Third, the signature overlap check is more expensive because it can not make use of pre-sorted signature lists.

PSC is inherently vulnerable to data skew, i.e., a non-uniform distribution of signatures. We therefore combine redundant pair elimination with our previous work on data skew handling [6] to achieve efficient MR programs. Our skew handling approach first executes a data analysis job to obtain global data statistics (e.g., cluster / block sizes) that is then used for skew handling in the actual PSC program. As a side-effect, we can replace each string-valued signature with its index in the list of all signatures to further speed up

Strategy	Pairs	Redundancy	Pairs completeness
1	$\approx 0.81 \cdot 10^9$	0%	0.75
1-2	$\approx 1.11 \cdot 10^9$	7.61%	0.90
1-3	$\approx 1.97 \cdot 10^9$	16,82%	0.98
1-4	$\approx 2.31 \cdot 10^9$	28,09%	≈ 1

Pass	Blocking keys
1	$\lfloor \log_2(\text{price}) \rfloor$
2	title.substr (0, 3)
3	category.substr (0, 10)
4	manufacturer

Figure 4: Resulting pairs completeness and degree of redundancy for different blocking strategies employing 1-4 passes (left) and signature generation for each pass (right).

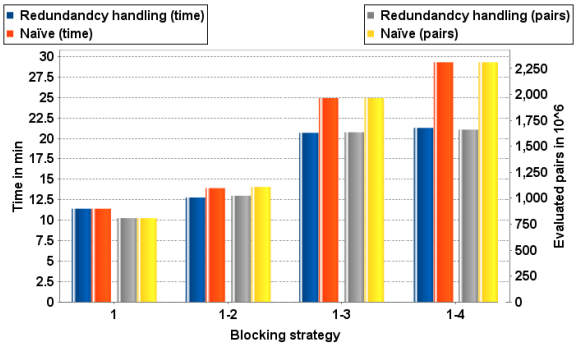

Figure 5: Execution time and number of evaluated pairs for all blocking strategies.

the overlap check (Algorithm 2). Unfortunately, our least common signature approach inherently introduces skew for the reduce phase. If a pair (o_1, o_2) shares several signatures, the pair will be processed for the smallest signature only. For all other (larger) signatures, the pair is considered redundant. In general, it is more likely that a pair is considered to be redundant for large key / signature values than for smaller values. We address this problem in future work.

Our approach for redundancy-free PSC is easy to integrate into existing MR-based PSC applications. As described, the basic implementation scheme sketched in Section 1 needs to be changed only slightly. The map output value is enriched by some additional information (i.e., the list of smaller signatures). Thus, the sorting, grouping, and repartitioning behavior as well as the key itself remain unchanged. The reduce function employs this additional metadata to eliminate redundant comparisons. We were able to employ our optimization technique into the work of [15] with a few code changes only. The authors published a patch file, provided by us, along with their source code (see `http://asterix.ics.uci.edu/fuzzyjoin/`).

5. EVALUATION

We evaluated our approach in the context of entity resolution with our prototype Dedoop [5]. We ran our experiments on Amazon EC2 using 20 worker instances of type *c1.medium* and a dedicated master instance of type *m1.small*. On each node, we set up Hadoop 0.20.2 with two map and reduce tasks running in parallel. We utilized a real-world datasets containing about 114,000 product offers. Our first experiment demonstrates the importance of utilizing multiple signatures per entity to improve match quality as well as the impact of the introduced redundancy on the overall run-time. In our second experiment, we control the degree of redundancy by using an artificial blocking function and analyze the resulting execution time with and without elimination of redundant pairs.

Number of signatures per object: We evaluated the full Cartesian product of the input data and compared the entity pairs by computing the trigram similarity on their product title. Two entities with a similarity $\geq$0.75 were regarded as matches. We then evaluated four different blocking strategies employing up to four blocking keys per entity. For each strategy, Figure 4 shows the achieved pairs completeness, i.e., the share of matching entity pairs from the evaluation of the Cartesian product that is retained despite the applied blocking. For instance, the first blocking key alone results in a completeness of only 0.75 (w.r.t. all entities pairs from the Cartesian product having a title similarity of at least 0.75), i.e., 25% of the matching pairs are not in any cluster. With additional blocking keys we can significantly improve match quality up to near-optimal pair completeness for four passes. At the same time, additional passes lead to a substantial degree of redundant pair comparisons of up to 28%. Figure 5 compares the observed execution times with and without redundancy avoidance. The redundancy-aware implementation clearly outperforms the naïve approach for all three multi-pass cases. In general, we observed run-time savings of the redundancy-aware implementation proportional to the cluster overlap for this experiment. The redundancy-free implementation could thus reduce execution times by about 8% for two passes and almost 30% for four passes.

Degree of redundancy: In our next experiment we study the robustness of our approach for systematically varied degrees of redundancy. We control the degree of redundancy by analyzing clusters with different degrees of overlap. Given a list of n entities, we initially partition the entities into c equally sized clusters of size $a = n/c$ by assigning entity $0 \leq i < n$ to cluster $\lfloor i/c \rfloor$. Hence, cluster $0 \leq i < c$ contains all entities in the interval $[i \cdot a,\ (i+1) \cdot a - 1]$. We fix the number of clusters and their lower bounds but shift the upper bound of each interval to the right to enlarge each cluster to size $a \leq s < n/2$. Thus, cluster $0 \leq i < c$ contains all entities in the interval $[i \cdot a, (i \cdot a + s - 1) \bmod n]$. Obviously, the cluster size s, controls the degree of cluster overlap and thereby the degree of redundant pairs.

Figure 6 compares the execution times for redundancy-free and naïve PSC of $n = 100{,}000$ entities partitioned into $c = 100$ clusters. We successively increase the initial cluster size by 1,000 entities up to $s = 10,000$. This results in a pair overlap between 25% for $s = 2{,}000$ and 81% for $s = 10{,}000$, respectively. While the execution time of the naïve implementation grows proportionally to the number of comparisons, the redundancy-free PSC strongly benefits from the saved comparisons and completes much faster although it achieves the same pair completeness. For the largest cluster size $s = 10,000$, the execution times are reduced by about a factor 4 when applying the proposed approach. Figure 6 shows that the execution time of the redundancy-free PSC grows slightly more than the number of pairs to evaluate. This is due to the overhead of the scheme since we still have to check for each pair in a cluster whether it has to be evaluated or whether it is redundant. However, the results show that this overhead is very small compared to the achieved savings by avoiding redundant comparisons.

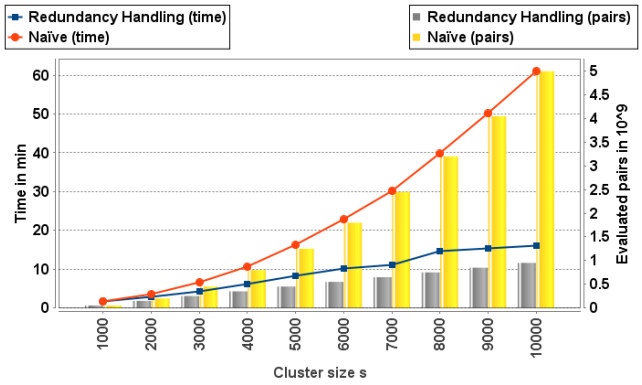

Figure 6: Execution time and number of evaluated pairs for different degrees of redundancy.

6. RELATED WORK

PSC is an important aspect of many data-intensive applications of different domains. To reduce the inherently quadratic complexity, a common optimization approach for PSC is to prune the set of all possible pairs to likely candidate pairs. Particularly for entity resolution, the literature proposes many so-called blocking techniques (see [2] for a survey). The realization of such blocking-based entity resolution with MR is relatively easy but uneven block sizes may require the use of load balancing strategies to effectively utilize all processing nodes [6]. The idea of pruning the set of all possible pairs to candidate pairs is applied by MR-based PSCs of other domains as well [1, 15].

In general, it is highly beneficial to assign objects to multiple clusters to be more robust against common errors like noisy, inconsistent, and missing attribute values as proposed in [9, 7, 13]. Surprisingly, only few studies considered the problem of redundant pair comparisons due to overlapping clusters. The first study we are aware of, investigated the problem in a non-distributed environment and proposed an approach called comparison propagation [12]. The key idea is that an object pair needs to be compared for a specific block only if their *least common block index* equals to the index of this block. This ensures that each entity pair is compared exactly once. The authors propose an algorithm to create a new set of semantically equivalent blocks without redundant comparisons. Each non-redundant comparison basically forms an individual block, resulting in a very large number of blocks with the cost of a quadratic space complexity. To overcome the high space requirements, the authors propose a complex preprocessing to reorganize and merge blocks. In a MR environment this step would require to run additional, preparatory MR jobs because the input data is distributed across several nodes and stored in a distributed file system. Another recent work proposes to chain a MR workflow consisting of two jobs in order to remove duplicate pairs from the match result that originate from overlapping blocks [9]. However, our approach does not require postprocessing the final PSC result.

7. SUMMARY

Efficient PSC implementations become feasible by the use of clustering to reduce the search space and the use of parallel processing based on a MapReduce framework. Robust implementations employ multiple clustering passes based on different signatures to cope with noisy, missing, and inconsistent attribute values. A side-effect of such approaches is the introduction of overlapping clusters leading to the redundant matching of the same object pairs. We therefore proposed a new and simple approach to eliminate such redundant pair comparisons within a MR-based parallel implementation. We implemented and evaluated the approach in our prototype Dedoop for MR-based entity resolution. The evaluation shows that our approach is able to significantly outperform the naïve PSC implementation and that it improves the performance for all degrees of redundancy.

Acknowledgments: We thank the anonymous reviewers for their valuable comments for improving this paper.

8. REFERENCES

[1] R. Baraglia, G. D. F. Morales, and C. Lucchese. Document Similarity Self-Join with MapReduce. In *ICDM*, 2010.

[2] P. Christen. A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication. *IEEE Trans. Knowl. Data Eng.*, 24(9), 2012.

[3] J. Ekanayake, T. Gunarathne, and J. Qiu. Cloud Technologies for Bioinformatics Applications. *IEEE Trans. Parallel Distrib. Syst.*, 22(6), 2011.

[4] T. Elsayed, J. J. Lin, and D. W. Oard. Pairwise Document Similarity in Large Collections with MapReduce. In *ACL (Short Papers)*, 2008.

[5] L. Kolb, A. Thor, and E. Rahm. Dedoop: Efficient Deduplication with Hadoop. *PVLDB*, 5(12), 2012.

[6] L. Kolb, A. Thor, and E. Rahm. Load Balancing for MapReduce-based Entity Resolution. In *ICDE*, 2012.

[7] L. Kolb, A. Thor, and E. Rahm. Multi-pass Sorted Neighborhood Blocking with MapReduce. *Computer Science - R&D*, 27(1), 2012.

[8] H. Köpcke and E. Rahm. Frameworks for entity matching: A comparison. *Data Knowl. Eng.*, 69(2), 2010.

[9] N. McNeill, H. Kardes, and A. Borthwick. Dynamic Record Blocking: Efficient Linking of Massive Databases in MapReduce. In *QDB*, 2012.

[10] M. Mendes and L. Sacks. Evaluating fuzzy clustering for relevance-based information access. In *IEEE FUZZ*, volume 1, 2003.

[11] C. Moretti, H. Bui, K. Hollingsworth, et al. All-Pairs: An Abstraction for Data-Intensive Computing on Campus Grids. *IEEE Trans. Parallel Distrib. Syst.*, 21(1), 2010.

[12] G. Papadakis, E. Ioannou, C. Niederée, et al. Eliminating the Redundancy in Blocking-based Entity Resolution Methods. In *JCDL*, 2011.

[13] G. Papadakis and W. Nejdl. Efficient Entity Resolution for Large Heterogeneous Information Spaces. In *ICDE Workshops*, 2011.

[14] M. C. Schatz. CloudBurst: highly sensitive read mapping with MapReduce. *Bioinformatics*, 25(11), 2009.

[15] R. Vernica, M. J. Carey, and C. Li. Efficient parallel set-similarity joins using MapReduce. In *Sigmod*, 2010.

[16] C. Xiao, W. Wang, X. Lin, et al. Efficient Similarity Joins for Near-Duplicate Detection. In *WWW*, 2008.