Distributed Privacy-Preserving Record linkage using Pivot-based Filter Techniques

Marcel Gladbach [1], Ziad Sehili [1], Thomas Kudraß [2], Peter Christen [3], Erhard Rahm [1]

[1] *Department of Computer Science, University of Leipzig, and ScaDS Dresden/Leipzig, Germany*
{gladbach,sehili,rahm}@informatik.uni-leipzig.de

[2] *Department of Computer Science and Mathematics, Leipzig University of Applied Science, Germany*
thomas.kudrass@htwk-leipzig.de

[3] *Research School of Computer Science, The Australian National University, Canberra, Australia*
peter.christen@anu.edu.au

***Abstract*—Privacy-preserving record linkage (PPRL) aims at linking person-related records from different data sources while protecting privacy. It is applied in medical research to link health data without revealing sensible person-related data. We propose and evaluate a new parallel PPRL approach based on Apache Flink that aims at high performance and scalability to large datasets. The approach supports a pivot-based filtering method for metric distance functions that saves many similarity computations. We describe our distributed approaches to determine pivots and pivot-based linkage. We also demonstrate the high efficiency of the approach for different datasets and configurations.**

I. Introduction

Record linkage or entity resolution aims at linking records that refer to the same real-world entity, such as persons or products. Typically there is a lack of global identifiers, therefore the linkage can only be achieved by comparing available quasi-identifiers (*QIs*), such as name, address or date of birth. However, in many cases, data owners are only willing or allowed to provide their data for such data integration if there is sufficient protection of sensitive information to ensure the privacy of persons, such as patients or customers.

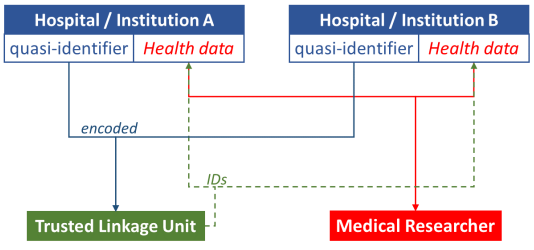


Figure 1. Medical application scenario for PPRL

Privacy-preserving record linkage (PPRL) is a promising approach to allow the integration and use of person-related data without revealing the identity of persons [1], [2]. For this purpose, the linkage of person-related records is based on encoded values of the *QIs* and the data needed for analysis (e.g., health data) is separated from these *QIs*. Figure 1 illustrates PPRL for two data sources (e.g., hospitals) and the use of a trusted unit to perform the linkage. This linkage unit only receives the encoded *QIs*, but not the health data relevant for analysis and medical research [3]. The latter is provided to the researchers without the person-specific *QIs* together with a linking ID (pseudonym) allowing the combination of medical data from different sources for improved data analysis and research. Those PPRL approaches can be applied in many areas, such as public health surveillance, crime detection, demographical studies and marketing analysis [4]. In recent years, it has primarily been used for medical research in different countries, as described in Section II.

PPRL is confronted with several challenges needing to be solved to ensure its practical applicability. In particular, a high degree of privacy has to be ensured by suitable encoding of sensitive data and organizational structures, such as the use of a trusted linkage unit. Despite the use of encoded *QIs*, PPRL must achieve a high linkage quality by avoiding false or missing matches. Furthermore, a high efficiency with fast linkage time and scalability to large data volumes are needed. A main problem for performance is the inherent quadratic complexity of the linkage problem when every record of the first source is compared with every record of the second source. For better efficiency, the number of comparisons can be reduced by adopting blocking or filtering approaches [4]. Furthermore, the linkage unit can perform PPRL in parallel on multiple processing nodes.

In this paper, we focus on improving the performance of PPRL by applying both filtering and parallel processing. We follow a pivot-based filtering approach for metric distance functions that has been shown to allow the elimination of many comparisons for centralized PPRL [5]. However, the centralized version of this approach is not sufficient for large datasets with millions of records as increasingly to be expected for applications such as in precision medicine. In order to deal with such large datasets, we additionally support a parallel PPRL for the pivot-based metric space approach based on the distributed processing framework Apache Flink [6].

Specifically, we make the following contributions:

- We propose parallel algorithms for both determining the pivots and the pivot-based matching process for different strategies to select pivots (Section III).

- We outline the implementation based on Apache Flink, a modern framework supporting efficient distributed in-memory processing (Section III).
- We comprehensively evaluate the performance and scalability of the proposed methods for different datasets and many configurations (Section IV).

In Section II, we start with discussing related work and outlining the pivot-based metric space approach for PPRL. After describing our distributed algorithms along with their implementation in Section III, we evaluate the approaches in Section IV, leading to a conclusion in Section V.

II. Related Work and Background

The use of PPRL for integrating several health-related databases is of increasing relevance for data analysis and research in medical applications, e.g., for epidemiological studies or adverse drug reaction studies [7]. Several medical use cases of PPRL are already reported. For example, Australian researchers linked data from Hospitals and Clinical Cancer Registries with data from Central Cancer Registers and the Bureau of Statistics in a privacy-preserving context to compare surgical treatment received by Aboriginal and non-Aboriginal people with lung cancer [8]. Further, long-term consequences of childhood cancer were analysed in Switzerland linking data from several cantonal and national registries using Bloom filters [9]. In Germany, pseudonymisation and PPRL services are used in several cooperative research projects involving multiple hospitals [10].

As surveyed in [2], [4], a large number of PPRL approaches has already been proposed to address research challenges such as improving security and privacy, linkage quality and scalability. For high efficiency and scalability it is important to avoid the quadratic complexity of the problem (comparing each record from one source with all records from the other source) by adopting blocking and filtering techniques [4]. In particular, pivot-based filtering strategies have shown to be very effective for centralized PPRL [5] and more efficient than a privacy-preserving version of PPJoin (called P4Join) filtering [11]. We will describe the pivot approach in Section II-B.

Parallel and distributed methods for PPRL have received very little attention so far. One previous PPRL study considered the use of MapReduce but in combination with LSH (locality sensitive hashing) as blocking method [12]. Parallel record linkage for unencoded data has also been studied primarily for MapReduce with one recent study for metric space distance functions [13]. However, the use of modern frameworks such as Apache Spark or Flink has not yet been considered for PPRL although it promises much better performance than applying MapReduce [4].

A. PPRL Setup

As in most previous PPRL studies, we follow a three-party protocol with two data owners and a trusted linkage unit [4] as illustrated in Figure 2. The data owners first exchange the exact parameters for encoding their data, in particular, which attributes and encoding functions to use. The encoded datasets are then sent to the linkage unit performing the linkage algorithm to identify similar records. The pairs of IDs of matching records are then sent back to the data owners, that can now link their sensitive data without revealing any personal information [4] [5].

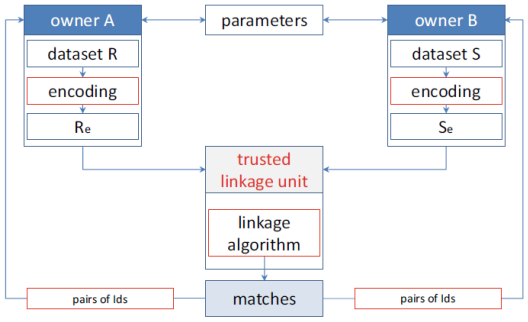


Figure 2. Three-party protocol for PPRL

For the encoding of the data we use Bloom filters [14] that are increasingly adopted for PPRL in real applications [3]. The attribute values of the records are tokenized into a set of n-grams, where multiple hash functions are applied to the n-grams by setting specific bits of a bit array with a fixed length to 1.

Acting as the linkage unit we receive the datasets as encoded bit arrays to find similar ones. In the following we refer to the first data source (usually the larger one) as *index data source* I with elements (Bloom filter) $i \in I$. The second data source is denoted as *query data source* Q with elements (Bloom filter) $q \in Q$.

B. Metric Space for PPRL

A metric space M is defined as the pair $M(X, d)$ of a set of data objects X and a distance function d, which has to satisfy several properties, in particular the triangle inequality: $\forall a, b, c \in X : d(a, c) \leq d(a, b) + d(b, c)$ [15], which can be utilized for filtering.

As distance function on Bloom filters we use the Hamming distance [15]:

$$d_h(a, b) = |a \vee b| - |a \wedge b| = a \; XOR \; b \tag{1}$$

It is especially applicable for bit arrays by applying bit-wise XOR to determine the number of differently set bit positions.

Typically in PPRL, a threshold $t \in [0, 1]$ is applied to define the minimal similarity two matching records have to meet. The similarity of two records can be calculated with the Jaccard similarity [15]

$$sim_j(a, b) = \frac{|a \wedge b|}{|a \vee b|} \tag{2}$$

Since we want to use the Hamming distance d_h to calculate distances in the PPRL process however, we have to define a similarity radius $rad(q)$ for elements $q \in Q$ converting the threshold t into a maximal Hamming distance. This can be achieved with the following relation between Jaccard similarity and Hamming distance $sim_j(x, q) \geq t \Leftrightarrow d_h(x, q) \leq$

$(|x|+|q|)(\frac{1-t}{1+t})$ and the length filter $|x| \leq \frac{|q|}{t}$. The resulting equation for the similarity radius is [5]

$$rad(q) = |q|\frac{1-t}{t} \tag{3}$$

while $|q|$ is the cardinality of q (the number of 1-bits). All records matching with q must have a distance lower or equal to $rad(q)$, i.e., they must lie in the circle around q with this radius.

To reduce the number of comparisons we select several elements from I as pivot elements $p \in P$, where $P \subset I$ is the set of all pivots. All other elements from I are assigned to their closest pivot. For each pivot p we obtain a set $set(p)$ of its assigned elements i with their distances $d_h(i,p)$ as well as the radius $rad(p)$ indicating the largest distance from the pivot to its elements.

We can now utilize the triangle inequality to filter distance comparisons between I and Q. First, each element $q \in Q$ is compared to all pivots $p \in I$ by calculating $d_h(p,q)$. q doesn't need to be compared to any $i \in set(p)$, if the first filter

$$d_h(p,q) > rad(q) + rad(p) \tag{4}$$

(*pivot filter*) applies, because the radii of p and q don't overlap. For the remaining pivots (as the example in Figure 3), we use the pre-calculated distances $d_h(i,p)$ for a second filter step (*distance filter*). If

$$d_h(p,q) - d_h(p,i) > rad(q) \tag{5}$$

then q and i cannot match and their comparison can be saved. Only for leftover candidate pairs of q and i, their distance $d_h(i,q)$ has to be calculated.

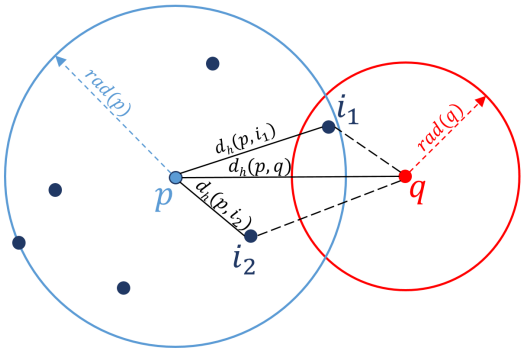


Figure 3. Utilization of the triangle inequality in Metric Space

C. Determining Pivots

The effectiveness of the pivot usage significantly depends on the number of pivots and how pivots are selected. In general there are only heuristics to find "good" pivots. A minimal number of overlaps between the pivot radii is considered to be a necessary requirement as it reduces the cases, where multiple records of Q have to be compared to multiple pivots p and their assigned elements. This indicates, that pivots on the "edge" of the metric space are preferable [16]. To find the pivots we apply two iterative state-of-the-art algorithms with runtime complexity $\mathbb{O}(|I| \cdot |P|)$:

1) Maximum Separation (ms): *ms* aims at maximizing the sum of distances between pivots. After picking the first pivot randomly the following ones are chosen by selecting the element having the largest sum of distances to all previously selected pivots [16].

2) Farthest-First-Traversal (fft): *fft* tries to select "corners" in the metric space as pivots. In each iteration, for every element the smallest distance to all previously selected pivots is calculated. The element for which this distance is the largest, is selected as the next pivot [16].

III. Distributed PPRL

Our distributed PPRL method using the pivot-based metric space approach works in two main phases, that are executed in parallel: preprocessing and matching. The preprocessing consists of several steps. First, *local* pivots on each partition of the index dataset are determined. From the union of the local pivots a *global* selection of the final pivots is performed. We consider the following strategies to determine pivots:
Local strategies:

- Random selection (*random*)
- Farthest-First-Traversal (*fft*)
- Maximum Separation (*ms*)

Global strategies:

- No global processing (*none*)
- Farthest-First-Traversal (*fft*)
- Maximum Separation (*ms*)

The assignment of records $i \in I$ to the chosen pivots p is also conducted in the preprocessing phase. In the matching phase for each query record $q \in Q$ the relevant pivots are determined with the pivot filter. Then, the actual matching of candidate pairs is performed after applying the distance filter.

For the implementation we use Apache Flink, a popular open-source framework for parallel data processing in Shared Nothing clusters [6]. It promises high performance, availability and accuracy for distributed data streaming applications as well as high throughput with low latency. The processing of data is handled as event-at-a-time by a distributed streaming data flow engine in the core. We use the Flink *DataSet API* for batch processing. It provides many powerful operators to perform different transformations on datasets that go beyond the simple MapReduce paradigm [6].

Figure 4 outlines the main steps of our Flink implementation for preprocessing on the index data source I (steps 1-4) and matching with the query data source Q (steps 5-6). The records (Bloom filters) of both sources are horizontally distributed among the N workers in equally sized partitions.

A. Preprocessing

In step 1, the local pivots P_L on each partition are determined with *MapPartition*. This operator allows us to apply an algorithm to all records of one partition and can return any number of records. Next, the global pivots P_G are determined (step 2). This is the only step in the process not running distributed. It uses the *GroupReduce* operator on the combined

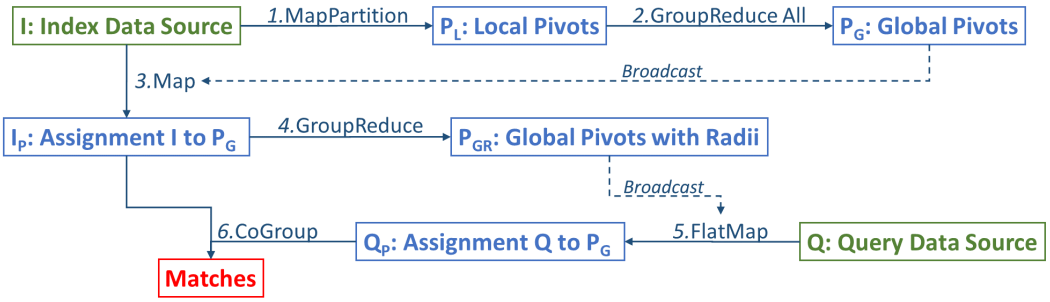


Figure 4. Flink-based PPRL processing (input datasets are in green; edges represent Flink operators)

dataset P_L, which is treated as one partition from which the global pivots are selected. The determined global pivots P_G are held in a distributed cache, accessible on each partition. Within Flink this set is made available as *Broadcast Variable* [17] for the *Map* operator in step 3. In this step, the index records i are assigned to their closest pivot p from P_G. For each record i, the pivot assignment (i, p) as well as the calculated distance $d_h(i, p)$ is stored as I_P. In step 4, we further determine the radius for each pivot by applying the *GroupReduce* operator to group the records per pivot p and determine the maximal distance $d_h(i, p)$ (using the previously calculated distances) for all assigned index records. We store the global pivots with their radii in P_{GR}.

B. Matching

In the matching phase, we determine for each query record q the global pivots that may contain match candidates. This is achieved with the *FlatMap* operator in step 5 applying the pivot filter (Equation 4) after determining q's similarity radius $rad(q)$ according to Equation 3. The result dataset Q_P contains the relevant query/pivot combinations (q, p) together with their distance $d_h(q, p)$ and the similarity radius $rad(q)$. A single record q might appear more than once in that set or even zero times, which already excludes it as possible match. Given the assignments I_P and Q_P, the records can now be distributed by the pivots, where each group consists of the pivot and its assigned index and query records. The previously executed grouping of I_P from step 4 can be used again. With the so-formed groups of index and query records we perform the final matching using the *CoGroup* operator in step 6 by first applying the distance filter (Equation 5) for each pair (i, q). For all candidate pairs remaining after the distance filter the distance $d_h(i, q)$ has to be calculated and compared with $rad(q)$ to find matching pairs.

Throughout the process we used the Hamming distance d_h for comparisons, because it can be calculated more efficiently for bit arrays than the Jaccard similarity. However, $rad(q)$ only gives us a upper bound regarding the given threshold t. Thus, we have to calculate the Jaccard similarity for all the possible matches after the final step and compare it to t. As our experiments have shown this is still the more effective and faster way for the overall process. All pairs satisfying the threshold t are returned within the match result.

IV. Evaluation

Our evaluation considers three different datasets with up to 8 million records and clusters with up to 16 workers. We investigate the influence of the different local and global pivot strategies as well as the scalability and speedup behaviour for different data and cluster sizes. We also compare the pivot-based approach with a baseline method using P4Join filtering.

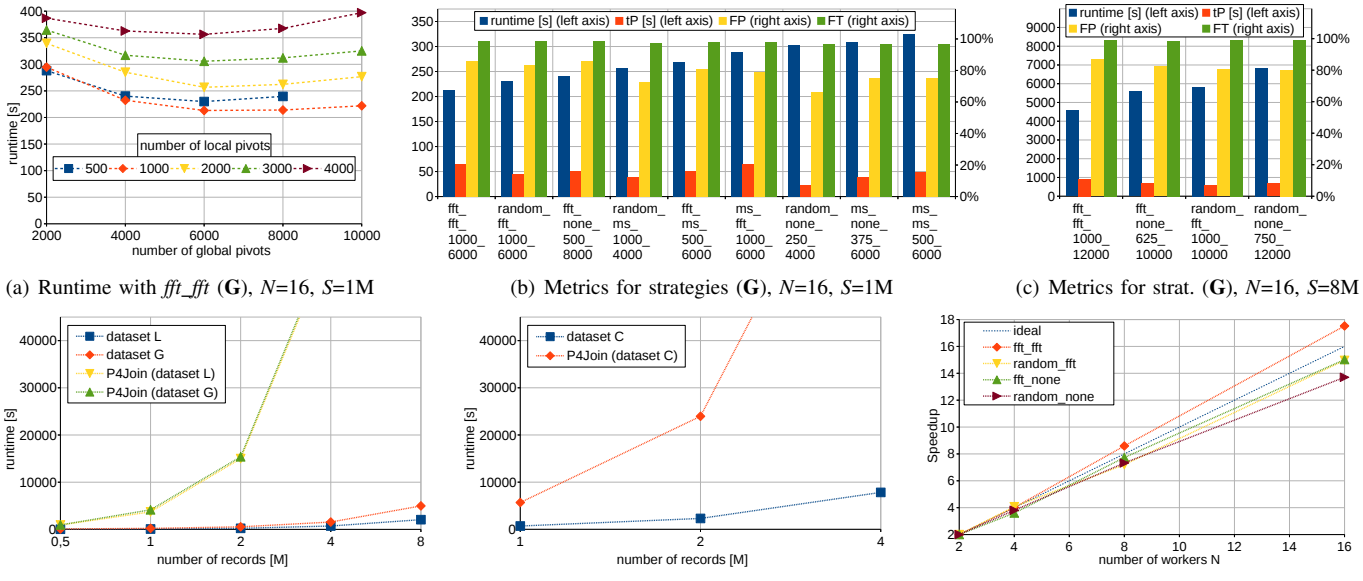
A. Experimental Setup and Metrics

Table I summarizes the main characteristics of the evaluation datasets. Due to the privacy context of this topic, real data from medical sources like hospitals is hard to obtain. Therefore we chose accessible real data as the third dataset **C**. It is generated from the voter registration data in North Carolina [18] using snapshots from different points of time. The other two datasets (**G** and **L**) are synthetically generated and corrupted using the GeCo Tool [19] with different numbers of records $S = n_I + n_Q$. Record creation is based on look-up files [20] for postcodes, cities and names from entire Germany (dataset **G**) or only from the region around Leipzig (dataset **L**). The latter dataset represents a more regional application scenario, e.g., for patients from close-by hospitals.

TABLE I
Dataset characteristics

	G	**L**	**C**
Tool	GeCo[19]		real-world data
Look-up Files / Origin	Germany	Leipzig	North Carolina
number of records S	0.5M to 8M		1M to 4M
Ratio $(n_I : n_Q)$	80:20		50:50
Errors per record	1		unknown
Duplicates in Q	100%		50%
Bloom Filter Length	1212		1097
chosen threshold t	0.88		0.85
recall (with t)	98.01%	97.90%	75.00%
precision (with t)	100.00%	100.00%	95.12%
F1-score (with t)	99.00%	98.94%	83,87%

The data is encoded based on trigrams that are mapped with 20 hash functions into Bloom filters of variable length (we use different Bloom filter sizes depending on the average number of trigrams as proposed by [21]). Since we focus on the evaluation of runtime and filter effectiveness, we apply a fixed similarity threshold leading to the best match quality (F1-score) for $S = 1$M. Table I also shows these thresholds



(a) Runtime with *fft_fft* (**G**), N=16, S=1M

(b) Metrics for strategies (**G**), N=16, S=1M

(c) Metrics for strat. (**G**), N=16, S=8M

(d) Scalability (**G** and **L**), N=16

(e) Scalability (**C**), N=16

(f) Speed-up for strategies (**G**), S=1M

Figure 5. Experimental results

and the resulting recall, precision and F1-score values. For the real dataset **C**, we achieved the lowest quality due to a higher degree of data errors. The linkage quality however only depends on the threshold t. It is not changed by the other parameters considered in the following.

The experiments are conducted on a cluster with 16 worker nodes, each equipped with 4 CPU-cores, 4 task slots and 4GB JVM Heap Memory using Hadoop, version 2.7.3 and Flink, version 1.2.1. To evaluate performance, we analyse the total runtime of the algorithm for different configurations as well as the preprocessing time t_P until step 4 is completed. To evaluate the effectiveness of the two filters, we determine which fraction of the number of comparisons they help to save compared to the maximal number of comparisons $n_I \cdot n_Q$ of the Cartesian product. The percentages F_P and F_T show how many comparisons are filtered out by the pivot filter alone (with Equation 4) and by both the pivot filter and the distance filter (after applying Equation 5), respectively ($F_P <= F_T$).

We consider the following configuration parameters:

- Number of workers N
- Local strategy s_L (*random*, *ms* or *fft*)
- Global strategy s_G (*none*, *ms* or *fft*)
- Number of local pivots n_L
- Number of global (overall) pivots n_P

In the following we use a certain notation for the chosen strategy and the number of pivots: $s_L\_s_G\_n_L\_n_P$ (for example *random_fft_2000_6000*). If not stated otherwise, we apply $N = 16$ and $S = 1$M for the number of workers and records.

B. Results

1) Number of Pivots: For each combination of local and global strategy, we looked at different numbers for local and global pivots. As an example, Figure 5(a) shows the runtime for strategy *fft_fft* on dataset **G** for different values of the number of global pivots n_P (x axis) and local pivots n_L (differently colored curves). For this strategy the optimal number of global pivots is 6000, no matter which n_L is picked. The combination with the best runtime is *1000_6000*, i.e. the 6000 final pivots are selected from 16000 (16·1000) local pivots. The results illustrate the trade-off to be made between a faster preprocessing with fewer pivots and a potentially stronger reduction of comparisons with more pivots.

2) Pivot Strategies: We now turn to the comparison between the different local and global pivot strategies. For this purpose we determine the best combinations of $n_L\_n_P$ with the shortest runtime for every possible strategy. These combinations are shown in Figure 5(b) for dataset **G** and $S = 1$M and in Figure 5(c) for the bigger number of records $S = 8$M. The combinations are ordered by their runtime from best to worst in the figures showing furthermore the time for preprocessing as well as the filter percentages F_P and F_T.

For both data sizes the best runtime and also the best filter percentages are achieved for the pivot strategy *fft_fft* and $n_L = 1000$. The pivot filter eliminates already 86-87% of the comparisons in these cases and the total filter percentage F_T reaches 98-99%. The preprocessing times are generally below 30% of the total runtime for the smaller dataset and below 20% for the bigger one indicating that for larger datasets a more sophisticated preprocessing for achieving bigger savings in match comparisons is a good strategy. The next best strategies for dataset **G** are *random_fft* or *fft_none* applying a cheap selection (random or none) for local pivots or global pivots, respectively. They have a lower preprocessing time but a slightly reduced (but still high) filter percentage F_P.

Interestingly, for both other datasets **L** and **C** ($S = 1$M) the *random_fft* combination had even the fastest runtime.

This shows, that the selection of the global pivots seems more important than local pivot selection and that a simple and fast $random$ strategy can be enough for determining local pivots. Although the best combinations differ somewhat between the different datasets, some results are consistent throughout all experiments. For example, all combinations with *fft* outperform the combinations with *ms*, especially on the local level, where *ms* is even worse than random selection.

3) Scalability: To evaluate scalability we analyse the performance of our parallel pivot-based PPRL approach for different data sizes (number of records) for both synthetic datasets **G** and **L** and the real dataset **C**. Furthermore, we compare the approach with a parallel implementation of the P4Join (a P4Join adoption for Bloom filters) [11] using length and prefix filtering. As Figure 5(d) shows for the synthetic datasets, the pivot space approach outperforms P4Join for all data sizes. P4Join also wasn't nearly as effective in filtering. For datasets **G** and **L** the length filter only reduced the total number of comparisons by 15% while the prefix filter had almost no effect. The reason are the relatively long bit vectors leading to long prefixes (about 100 bits) and thus a low probability, that prefixes do not overlap.

In **G** the records (bit vectors) are on average relatively far away from each other (large d_h). This leads to large pivot radii $rad(p)$ and thus to lower values for $F_P = 83\%$ and $F_T = 98\%$ for $S = 8$M. The records of **L** are more similar and therefore closer in average leading to $F_P = 94\%$ and $F_T = 99\%$ for 8M records. Accordingly, the runtime for those two datasets differs for all data sizes by roughly a factor of 2.

Figure 5(e) shows the scale-up for dataset **C**. With this dataset we observed a lower filter effectiveness as for the synthetic datasets. For $S = 1$M the numbers are $F_P = 40\%$, $F_T = 90\%$ for the fastest combination (*random_fft*) and $F_P = 54\%$, $F_T = 89\%$ for *fft_fft*. The resulting longer runtime is also influenced by the different ratio $n_I : n_Q$ (50:50), because the query dataset Q is much larger so that its assignment to the pivot and the matching takes much longer. However, our method still outperforms P4Join by far, which only filters about 13% of comparisons.

4) Speedup: We finally evaluate speedup for clusters sizes between 2 and 16 workers. As can be seen in Figure 5(f), for dataset **G** speedup is nearly linear for several pivot strategies (the graphs for the other datasets look very similar). For some strategies we even achieve a speedup better than linear (e.g., *fft_fft*). The reason is a long runtime for the cases with lower N, especially for $N = 2$, where a higher number of local pivots n_L was needed resulting in increased preprocessing times.

V. Conclusion

We proposed and evaluated a new distributed pivot-based PPRL method for metric distances. For implementation we utilized Apache Flink as state-of-the-art distributed processing framework. The experimental evaluation for different datasets showed that the farthest-first-traversal (fft) algorithm to find global pivots in combination with a fft or random strategy to find local pivots outperforms other strategies due to good filtering effects. The parallel approach also showed good scalability to larger data sizes and excellent speedup. For future work, we plan to investigate further distributed PPRL approaches and make them available in a toolbox for use in applications and for a comparative evaluation. Furthermore we cooperate with a consortium of university hospitals to build a PPRL service.

References

[1] R. Schnell, T. Bachteler, and J. Reiher, "Privacy-preserving record linkage using Bloom filters," *BMC Med. Inf. & Decision Making*, vol. 9, p. 41, 2009.

[2] R. Hall and S. Fienberg, "Privacy-Preserving Record Linkage," in *Privacy in statistical databases*, 2010, pp. 269–283.

[3] J. Boyd, S. Randall, and A. Ferrante, "Application of Privacy-Preserving Techniques in Operational Record Linkage Centres," in *Medical Data Privacy Handbook*, A. Gkoulalas-Divanis and G. Loukides, Eds. Springer, 2015.

[4] D. Vatsalan, Z. Sehili, P. Christen, and E. Rahm, "Privacy-Preserving Record Linkage for Big Data: Current Approaches and Research Challenges," in *Handbook of Big Data Technologies*, A. Zomaya and S. Sakr, Eds. Springer, 2017.

[5] Z. Sehili and E. Rahm, "Speeding up Privacy Preserving Record Linkage for Metric Space Similarity Measures," in *Datenbank Spektrum*, vol. 16, 2016, pp. 227–236.

[6] The Apache Software Foundation. (2017) Apache Flink. [Online]. Available: https://flink.apache.org/

[7] B. A. Malin, K. E. Emam, and C. M. OKeefe., "Biomedical data privacy: problems, perspectives, and recent advances," in *JAMIA*, vol. 20, 2013, pp. 2–6.

[8] A. Gibberd, R. Supramaniam, A. Dillon, B. K. Armstrong, and D. L. OConnel, "Lung cancer treatment and mortality for Aboriginal people in New South Wales, Australia: results from a population-based record linkage study and medical record audit," in *BMC Cancer*, 2016.

[9] C. E. Kuehni, C. S. Rueegg, G. Michel, C. E. Rebholz, M.-P. F. Strippoli, F. K. Niggli, M. Egger, and N. X. von der Weid, "Cohort Profile: The Swiss Childhood Cancer Survivor Study," in *Int. Journal of Epidemiology*, 2012.

[10] M. Lablans, A. Borg, and F. Ückert. (2017) Mainzelliste. [Online]. Available: http://www.unimedizin-mainz.de/imbei/informatik/opensource/mainzelliste.html

[11] Z. Sehili, L. Kolb, C. Borgs, R. Schnell, and E. Rahm, "Privacy Preserving Record Linkage with PPJoin," in *Proc. 16th BTW*, 2015.

[12] D. Karapiperis and V. S. Verykios, "A distributed near-optimal LSH-based framework for privacy-preserving record linkage," in *Comput. Sci. Inf. Syst.*, vol. 11, 2014, pp. 745–763.

[13] G. Chen, K. Yang, L. Chen, Y. Gao, B. Zheng, and C. Chen, "Metric similarity joins using MapReduce," in *IEEE TKDE*, vol. 29, 2017, pp. 656–669.

[14] R.Schnell, T. Bachteler, and J. Reiher, "A Novel Error-Tolerant Anonymous Linking Code," German Record Linkage Center, Duisburg, Tech. Rep. WP-GRLC-2011-02, 2011.

[15] P. Zezula, G. Amato, V. Dohnal, and M. Batko, *Similarity Search: The Metric Space Approach*. Springer, 2006.

[16] R. Mao, P. Zhang, X. Li, X. Liu, and M. Lu, "Pivot selection for metric-space indexing," in *Int. J. Machine Learning & Cybernetics*, vol. 7, 2016, pp. 311–323.

[17] The Apache Software Foundation. (2017) Flink Documentation Version 1.2. [Online]. Available: https://ci.apache.org/projects/flink/flink-docs-release-1.2/

[18] North Carolina State Board of Elections & Ethics Enforcement. (2017). [Online]. Available: http://www.ncsbe.gov/

[19] P. Christen and D. Vatsalan, "Flexible and Extensible Generation and Corruption of Personal Data," in *Proc. 22nd ACM CIKM*, 2013, pp. 1165–1168.

[20] Statistisches Bundesamt. (2017) Zensus 2011. [Online]. Available: https://www.zensus2011.de/

[21] E. A. Durham, M. Kantarcioglu, Y. Xue, C. Toth, M. Kuzu, and B. Malin, "Composite Bloom Filters for Secure Record Linkage," in *IEEE TKDE*, vol. 26, 2014, pp. 2956–2968.