Achieving k-anonymity of a large-scale database in a distributed memory environment

Arsème Vadèle[1], Djeufack Nanfack[2], Gaël Le Mahec[2], Gilles Dequen[2], Vianney Kengne Tchendji[3], and Jerry Lacmou Zeutouo[4]

[1]Affiliation not available
[2]MIS research unit, University of Picardie Jules Verne)
[3]Department of Mathematics and Computer Science, University of Dschang
[4]The joint project-team Avalon INRIA, LIP, ENS-Lyon

January 25, 2024

Abstract

The k-anonymity problem introduced by Samarati and Sweeney in 1998, guarantees that it is impossible to distinguish user data from at least (k - 1) others in the same database. The methods used to achieve k-anonymity result in an information loss as the data in the database is modified, making it less accurate through a process of generalization or micro-aggregation of the stored data. Mauger et al. proposed a $O(n^2)$-time sequential algorithm that gives good results while minimizing the information loss using their designed metrics. However, their solution is very time-consuming and therefore not suitable for large-scale databases. In this paper, we tackle this problem using parallelism. We propose three coarse-grained parallel algorithms to solve the k-anonymity problem. The first is the straightforward algorithm that runs in $O(n^2/p)$ execution time with O(n) communication rounds, where n is the number of lines in the database and p is the number of processors. The second runs in $O(n^2/p^2)$ execution time with $O^2(p)$ communication rounds. The third runs in $O(n^2/p\log2(p))$ execution time with $O(np/(\log2(p))^2)$ communications rounds. For the latter two algorithms, we introduce the concept of data reorganization to minimize the information loss when data are partitioned. Experimental results show that for a database of size n = 10ˆ6 , p = 2ˆ7 , and k = 10ˆ2 , second, and third parallel algorithms are respectively 1127.59× and 41.13× faster than the sequential algorithm while achieving anonymity with 4.03% and 2.62% information loss.

Achieving k-anonymity of a large-scale database in a distributed memory environment

Arsème Vadèle Djeufack Nanfack*, Gaël Le Mahec*, Gilles Dequen*, Vianney Kengne Tchendji† and Jerry Lacmou Zeutouo‡

* *MIS research unit (University of Picardie Jules Verne)*, Amiens, France
Email: {arseme.djeufack.nanfack, gael.le.mahec, gilles.dequen}@u-picardie.fr

† *Department of Mathematics and Computer Science* , *University of Dschang* , Dschang, Cameroon
Email: vianneykengne@yahoo.fr

‡ *The joint project-team Avalon INRIA, LIP, ENS-Lyon*, Lyon, France
Email: jerry.lacmou-zeutouo@inria.fr

***Abstract*—The k-anonymity problem introduced by Samarati and Sweeney in 1998, guarantees that it is impossible to distinguish user data from at least $(k-1)$ others in the same database. The methods used to achieve k-anonymity result in an information loss as the data in the database is modified, making it less accurate through a process of generalization or micro-aggregation of the stored data. Mauger et al. proposed a $\mathcal{O}(n^2)$-time sequential algorithm that gives good results while minimizing the information loss using their designed metrics. However, their solution is very time-consuming and therefore not suitable for large-scale databases. In this paper, we tackle this problem using parallelism. We propose three coarse-grained parallel algorithms to solve the k-anonymity problem. The first is the straightforward algorithm that runs in $\mathcal{O}\left(n^2/p\right)$ execution time with $\mathcal{O}\left(n\right)$ communication rounds, where n is the number of lines in the database and p is the number of processors. The second runs in $\mathcal{O}\left(n^2/p^2\right)$ execution time with $\mathcal{O}\left(p\right)$ communication rounds. The third runs in $\mathcal{O}\left(n^2/p\log_2(p)\right)$ execution time with $\mathcal{O}\left(np/\left(\log_2(p)\right)^2\right)$ communications rounds. For the latter two algorithms, we introduce the concept of data reorganization to minimize the information loss when data are partitioned. Experimental results show that for a database of size $n = 10^6$, $p = 2^7$, and $k = 10^2$, second, and third parallel algorithms are respectively $1127.59\times$ and $41.13\times$ faster than the sequential algorithm while achieving anonymity with 4.03% and 2.62% information loss.**

***Index Terms*—k-anonymity, information loss, privacy-preserving data publishing, parallel computing, coarse-grained multicomputer**

I. Introduction

THE masses of data collected today in various domains, especially in health, banking, and insurance, to name a few, are intended to be published for research purposes, for example. Among these data, some are private (referred to as sensitive data), which implies that they must be preserved when published. However, these data are subject to several risks when they are published. In particular, identity disclosure, attribute disclosure, inference disclosure, as well as membership disclosure [1]. Different methods have been proposed to address these different attacks. l-diversity and t-closeness have been introduced to deal with attribute disclosure attacks, as well as several other methods. In their work, Li et al. [2] addressed inference attacks, where attackers use a derived distribution from the published data. Among the proposed methods, some can address multiple attacks at once. Samarati and Sweeney [3] introduced k-anonymity for identity disclosure, and several variants have been developed in recent years [1]. In the rest of this paper, the focus is on k-anonymity.

The solutions proposed in the literature vary in their ability to preserve the data utility [4]–[12]. However, it is important to note that optimizing data utility during the k-anonymity process is an NP-hard problem [6], [9]. As a result, striking the right balance between information loss and computational cost becomes critical, especially in scenarios where the volume of data to be anonymized continues to grow over time. Parallelism serves as a solution to reduce the execution time of computationally intensive applications.

Over the past two decades, the landscape of parallel computing has been marked by the emergence of various parallel models, each aiming to harness the power of distributed systems architecture. Notable among these models are MapReduce [13], LogP [14], Bulk Synchronous Parallel (BSP) [15], and Coarse-Grained Multicomputer (CGM) [16]. The diversification of these models has presented a formidable challenge in crafting parallel algorithms for intricate problems such as recommender systems, optimal binary search trees, and minimum cost parenthesizing [17]–[19]. Among these models, the CGM model stands out as an apt choice for contemporary parallel systems comprised of distributed computing nodes, offering a platform for the creation of algorithms that are both efficient and portable.

The CGM model consists of computation rounds, in which individual computational units execute sequential algorithms on different subproblems, interspersed with communication rounds that allow data exchange between computational units. This distinctive approach has attracted attention for its ability to strike a balance between computation and communication, making it conducive to parallel algorithm design.

The focus of this paper is to address the challenge of parallelizing the k-anonymity problem within the confines of the CGM model. The aim is to exploit its strengths to efficiently solve this problem in a distributed memory environment.

In the following, a database is typically structured as a table, comprising rows referred to as records or tuples, and

columns known as attributes. There are three distinct types of attributes: *identifiers* that uniquely index users from the database, *quasi-identifiers* that do not provide any information about users, but which when cross-referenced with external data sources (such as electoral lists) allow users to be identified (for example postal code, gender, etc.). Finally, the *sensitive attributes* do not give any information about user identity (e.g. salary, medical illness, etc.). This last attribute type is usually the reason one wants to anonymize the database. It is important to clarify that this paper exclusively focuses on quasi-identifier attributes.

A. Related works

Anonymity can be achieved through various techniques, including non-perturbative techniques like global recoding, generalization, local recoding, suppression, and clustering, as well as perturbative methods such as noise addition, data swapping, micro-aggregation, and de-associative techniques [1] like bucketization [20], [21], angelization, and anatomization. To maintain clarity and focus, we will primarily explore approaches based on generalization in the following discussions. Generalization involves replacing specific values with more general or less precise ones, thereby reducing the level of detail in the data to ensure anonymity.

Global recoding-based approaches involve recoding records across the entire dataset by grouping specific categories of values together to create more generalized categories [4], [22], [23]. This method allows for the creation of more general representations of the data. On the other hand, local recoding-based approaches map attribute values to generalization values based on their respective groups [24]. In these approaches, the same attribute values may be generalized differently if they belong to different groups. Both global and local recoding-based approaches have the advantage of preserving good data utility by maintaining a higher level of detail in the anonymized data. However, the main drawback of these approaches is the significant computational cost associated with the recoding process. The computational complexity and time required for recoding the data can pose challenges, especially when dealing with large datasets.

Clustering-based approaches draw inspiration from the field of clustering research [9]–[12], [25], [26], such as the widely known k-means algorithm. In these approaches, clusters are referred to as equivalence classes, which consist of similar records based on a specified similarity criterion. Clustering-based methods offer reduced computation time due to the quadratic nature of their algorithms. However, they generally exhibit weaker data utility compared to global recoding techniques. For example, the GkAA algorithm employs a greedy heuristic based on a local choice approach [11]. It identifies suitable records that best fit into the same equivalence class, using a similarity measure based on calculated distances between records. It optimizes an information loss metric by considering attribute generalization hierarchies.

Sang et al. [10] proposed the $GCCG$ algorithm for achieving k-anonymity and demonstrated its parallelizability to expedite the anonymization process. They proposed a shared-memory parallel algorithm based on their sequential solution. Their experimental results showed that the parallel algorithm is $10.5\times$ faster than the sequential algorithm with four threads. However, as the number of threads increased, the parallel approach exhibited more significant data loss compared to the sequential one.

It is important to note that clustering-based approaches often rely on greedy algorithms that do not guarantee optimality in terms of data utility in the anonymized database. Nevertheless, these approaches offer more flexibility for execution on distributed systems that handle larger datasets.

There has been extensive research conducted on distributed privacy preserving data publishing over the last twenty years [27]–[31]. Within this domain, the primary challenges revolve around data partitioning and aggregation strategies. Researchers commonly assume that, data is distributed across multiple parties and needs to be combined to achieve anonymization for the global database, which is considered as the collective union of all parties' databases.

Wei and Chris [27] proposed an approach to anonymize a database by vertically partitioning it into subsets of quasi-identifier attributes from two different homogeneous sources. Their focus lies within the realm of IoT network security, where the different sources should not be able to decrypt data they don't have access to. Each source (party) performs generalizations on their respective data, and then the tables are joined based on well-defined comparison criteria. In a different study, Jurczyk and Xiong [28] introduced the first framework for horizontal database partitioning. This involves partitioning a dataset into multiple subsets with the same attributes but different records. However, Mohammed et al. [29] demonstrated that applying k-anonymization algorithms to different subsets of a database, viewed as a horizontally partitioned database in a distributed memory environment, leads to significant information loss compared to a centralized environment. In this scenario, all parties have their own data with the same attributes but different records (similar to [28]). Each party applies anonymization techniques to their data, and the anonymized data is shared with a designated party for aggregation. The resulting joined anonymized dataset is then published.

In 2021, Wook [30] proposed an approach for anonymizing data resulting from an SQL query on a database. In this approach, an SQL query is sent to a central server (the master node), which executes the query on relevant database servers (slave nodes). Each slave node returns its result to the master node for aggregation, and the master node anonymizes the data before returning it. The author acknowledged that achieving k-anonymity can be computationally expensive and time-consuming, especially with large datasets containing millions of records. To address this issue, the author leveraged the computational power of each slave node to anonymize data locally and then returned the anonymized result to the central server. The central server aggregates the different results and returns the anonymized data. The resulting table may not initially satisfy the k-anonymity constraint during result aggregation, so the anonymization process must be repeated until a k-anonymous table is obtained.

While these distributed-based approaches offer advantages,

they still face computational cost challenges. The anonymization algorithms used by each party do not effectively utilize the computational power of the processing units. This becomes problematic when parties have large amounts of data to process, making it difficult to handle even with quadratic algorithmic approaches.

In a later study conducted in 2022, researchers explored an approach to scalable data anonymization, as detailed in [31]. This approach was designed to address the challenges of anonymizing large datasets within a distributed memory environment, specifically utilizing Apache Hadoop. The foundation of their work stemmed from an extension of the well-established centralized Mondrian algorithm, as outlined in the original Mondrian paper [32]. The primary focus of their research revolved around optimizing computational efficiency through two key partitioning strategies. The first, known as quantile-based partitioning, showcased superior performance, especially when the number of available computing workers was not a power of 2. However, a limitation emerged as this strategy capped the number of partitions at the attribute with the largest domain size. On the other hand, the second strategy, termed multi-dimensional partitioning, demonstrated the potential to yield an unbalanced workload distribution among workers, potentially resulting in double the computational burden for some. Consequently, the efficiency of their approach hinged strongly on the ability to decompose the number of workers into a power of 2. The study's findings were promising, indicating substantial computation time savings ranging from 28% to 98% when applied to a dataset consisting of 1,000,000 records for various values of k (5, 10, and 20). These results were especially evident when the approach was executed with worker counts ranging from 2 to 12. Notably, the quantile-based partitioning strategy incurred only a slight difference in terms of information loss compared to the centralized Mondrian algorithm.

B. Our contribution

In the realm of distributed memory environments, the partitioning of databases can lead to a notable loss of information. This challenge becomes particularly pronounced as the number of processors (or workers) and the value of k (the anonymity constraint) increase, or when dealing with relatively small datasets. The earlier study conducted and detailed in [31] was restricted to a mere 12 workers. Consequently, this limitation failed to provide a comprehensive view of the information loss, particularly for higher k values, such as $k = 100$, resulting from excessive dataset partitioning. Moreover, the requirement for the number of workers to be a power of 2, as identified in previous work, imposes constraints on solution efficiency. In this paper, our primary focus is to tackle the challenges associated with mitigating information loss and optimizing computation time while achieving k-anonymity within a distributed memory environment. To address these concerns, the contributions proposed in this paper are twofold:

1) We introduce *data reorganization* as a preprocessing task, which involves pre-constructing equivalence classes to arrange records with higher similarity in close proximity. This objective is achieved through the application of a stable sorting method. We explicitly demonstrate the influence of the sorting order of quasi-identifier attributes on the partitioning procedure. By considering the generalization hierarchy levels of quasi-identifiers, we derive a generalized optimal order for data reorganization. Notably, this strategy proves effective in maintaining data utility within a distributed memory environment for high values of p and k.
2) We propose three coarse-grained parallel algorithms based on two partitioning strategies. The initial partitioning strategy focuses on identifying pairs of equivalence classes that can be iteratively merged until the desired k-anonymity level is achieved. This approach serves as the basis for the design of the first parallel algorithm, called PGkAA. However, the scalability of the PGkAA algorithm becomes limited as the database size increases. To overcome this issue, we propose a second partitioning approach. This approach claims that the process of database anonymization can be streamlined by anonymizing multiple partial databases extracted from the main database. We consider a scenario where a database T can be viewed as a union of p sub-databases ($T = \bigcup_i^p T_i$) where each couple (T_i, T_j) have exactly same attributes, with the assumption that these sub-databases either remain disjoint or have overlapping properties. This partitioning scheme forms the basis for the formulation of two advanced parallel algorithms, called PGPkAA and H-PGPkAA. The PGPkAA algorithm stands as a testament to the reduction in execution time during the anonymization process as the number of processors increases. Conversely, the H-PGPkAA algorithm deftly strikes an optimal equilibrium between the execution time incurred and the information loss that accompanies the anonymization process.

The PGPkAA and H-PGPkAA algorithms reorganize the data during a preprocessing phase. Experimental results showed that data reorganization mitigates information loss while achieving k-anonymity in a distributed memory environment.

The remainder of this paper is organized as follows. Section II provides an overview of the k-anonymity concept. Section III describes the reorganization concept and our parallel algorithms. Section IV outlines experimental results. Finally, Section V concludes the paper.

II. Privacy preserving and k-anonymity

Sweeney [33] showed how the identity of the Massachusetts state government could be found just by crossing the pseudonymized medical data of an insurance organization and the electoral list of Cambridge (USA). Samarati [4] introduced the k-anonymity concept which guarantees that it is impossible to distinguish an individual data in the database from at least $(k-1)$ others.

Fig. 1 shows an example of a database inspired by the Adult database [34]. First and last names are the *identifiers*, age, gender, workclass, and education are the *quasi-identifiers* and the income is the *sensible* value.

Identifiers		*Quasi-identifiers*				*Sensible*
First name	**Last name**	**Age**	**Gender**	**Workclass**	**Education**	**Income**
John	Doe	39	M	State-gov	Bachelors	=50K
Josef	Novák	32	M	Private	Masters	<50K
Navn	Navnesen	18	M	Never-worked	11th	=50K
Jan	Jansen	75	M	Federal-gov	Doctorate	=50K
Marie	Dupont	38	F	Private	Bachelors	<50K
Erika	Mustermann	22	F	State-gov	Doctorate	=50K
Kari	Nordmann	65	F	Federal-gov	Masters	<50K
Anna	Kowalska	39	F	Private	11th	=50K

Fig. 1: Example of a database

A. Data generalization

Some of the approaches used to achieve k-anonymity, use data generalization and suppression, making data less precise and leading to information loss. In clustering-based approaches, k-anonymity is considered a partitioning problem that consists of grouping objects with similar characteristics based on specific criteria. Thus, to achieve k-anonymity, we group the lines (records) of the database in clusters of equivalence classes such that all records have the same values according to quasi-identifiers. So all equivalence classes have at least k records.

Fig. 2a displays the generalized values for the quasi-identifier *age*. Here, we have grouped ages into distinct age categories. Fig. 2b highlights the possible generalizations for the education levels; the "*" symbol represents any education level and denotes also the highest level of generalization.

It is crucial to note that the choice of how to generalize a value can significantly impact the data's utility once the database has been made k-anonymous. Indeed the more the value is generalized, such as "*" or "[16-75] age group", the more information loss increases by decreasing the precision.

B. Information loss metrics

Data utility of a k-anonymous database can be measured with information loss metrics [11] or according to the efficiency obtained during the training of an AI model processed with the k-anonymous database [35]. This last evaluation is out of the scope of this paper. Indeed, if such efficiency measures can be very useful, they are also dependent on the context of the use of the data and the objectives sought (inference, generation, etc.). We focus on general information loss metrics that, in our sense, are more suitable for privacy-preserving data publication without having an idea of the end use of the data.

To evaluate the information loss rate, we use metrics based on generalization cost functions. These functions compute the cost of merging two rows as the sum of the generalization costs of their quasi-identifiers to make them identical as follows: If there is a direct path between two nodes n and m of the hierarchy then for a metric μ, $cost(n,m) = \mu(n,m)$. If there is no direct path from node n to node m, we determine the lowest common ancestor ($LCA\,(n,m)$[1]), then for a metric μ we find $cost(n,m) = \mu\,(n, LCA\,(n,m))$. Note that $cost(n,m)$ is generally not equal to $cost(m,n)$ because the path from one node to the LCA is generally not equal to the cost from the other node. Moreover, when there is a direct path from n to m, $cost(n,m) = cost(n, LCA\,(n,m))$. To determine all possible generalizations, these values are represented in matrices that completely define the *cost metric* for each attribute. We rely on these matrices to find a k-anonymous version of the database that minimizes information loss. Fig. 3 shows the corresponding cost matrices for the generalization hierarchies presented in Fig. 4.

Such a generalization metric can be arbitrarily defined (e.g. one can choose that generalizing a PhD in "Professional Degree" must cost more than generalizing a Master degree in the same value). However, defining such generalization costs can be very time-consuming or even impractical when the attribute can take many values. Many information loss metrics have been proposed over time to define automatically the generalization costs of semantic hierarchies. Moreover, the choice of these costs can lead to very different versions of the same database, with more or less information loss. In their work [11], Mauger et al. evaluated information loss for seven metrics (Distorsion [7], NCP [8], Total [9], LLM, NLLM, WLLM, NWLLM [11]) for large range of values of k (from 2 up to 15,000 for a dataset of 30,162 lines). Their results show that the NLLM metric is well suited to limit information loss when used to k-anonymize a database. We used it for our later experiments, but any other metric could easily be used by simply changing the cost matrices.

For an information loss metric μ, the data utility of a k-anonymous version of the database T is determined by the following relation [38] :

$$A_\mu\,(T) = \frac{\mu\,(T)}{\mu\,(T^*)} \times 100 \tag{1}$$

where T^* represents the k-anonymous database with maximum generalization (where all information has been lost). Considering such a metric, finding the best version of a k-anonymous database (i.e. using the minimum generalization possible) is NP-hard [9].

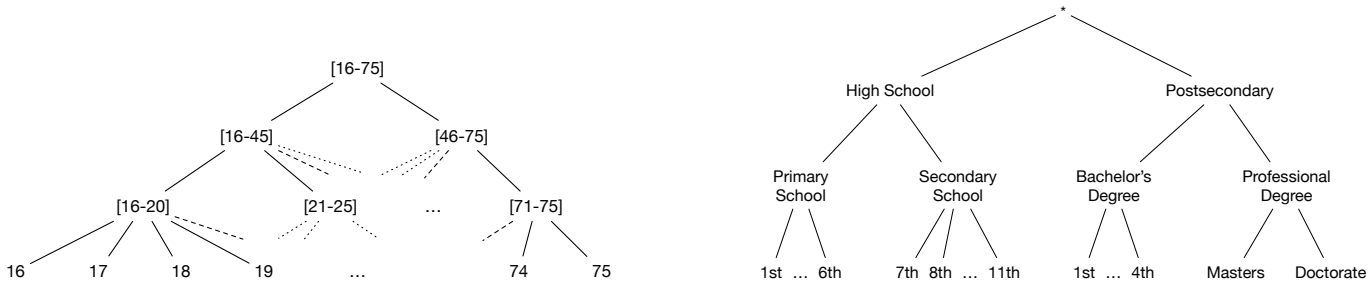
C. Sequential algorithm of Mauger et al. [38]

To minimize information loss, the authors of [11] adopted a strategy of merging two equivalence classes with the least merge cost, effectively seeking the pair of classes whose merger minimizes information loss. To achieve this objective, they introduced an algorithm that proceeds as follows: it initially selects a class, denoted as C_{small}, from the set of the smallest classes. Subsequently, within the remaining equivalence classes, the algorithm identifies the class whose merging incurs the minimal cost. These two classes are then merged, resulting in the formation of a new class. Throughout this process, it is essential to ensure that each equivalence class maintains a size of at least k. Algorithm 2 succinctly outlines the procedural steps undertaken within the search function for the class with the best merging cost with C_{small}.

Let T be a table of n records and $\mathcal{C}(T)$ the set of equivalence classes composed of r equivalence classes.

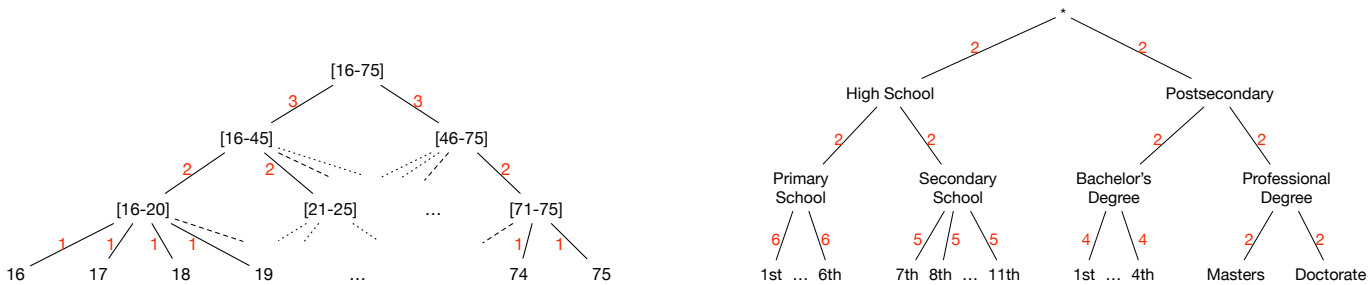
Theorem 1. *The GkAA algorithm runs in $O\left(n^2\right)$ time.*

[1]It represents the common ancestor of the two nodes which is the far from the root [36].



(a) *Age* (b) *Education*

Fig. 2: Generalization hierarchy



(a) *Age* (b) *Education*

Fig. 3: Weighted Generalization hierarchy [37]

$$
\begin{array}{c}
 \\ 16 \\ 17 \\ \cdots \\ 75 \\ [16-20] \\ \vdots \\ [16-75]
\end{array}
\begin{array}{c}
\begin{array}{ccccccc} 16 & 17 & \cdots & 75 & [16-20] & \cdots & [16-75] \end{array} \\
\begin{pmatrix}
0 & 1 & \cdots & 6 & 6 & \cdots & 6 \\
1 & 0 & & 6 & 6 & & 6 \\
\vdots & & \ddots & \vdots & & & \vdots \\
6 & 6 & \cdots & 0 & 6 & \cdots & 6 \\
0 & 0 & & 0 & 0 & & 3 \\
\vdots & & & \vdots & & \ddots & \vdots \\
0 & 0 & \cdots & 0 & 0 & \cdots & 0
\end{pmatrix}
\end{array}
$$

(a) *Age*

$$
\begin{array}{c}
 \\ 1st \\ 2nd \\ \vdots \\ Pr. \\ HS \\ \vdots \\ *
\end{array}
\begin{array}{c}
\begin{array}{ccccccccccc} 1st & \cdots & 7th & \cdots & M & D & Pr. & \cdots & HS & \cdots & * \end{array} \\
\begin{pmatrix}
0 & \cdots & 8 & \cdots & 10 & 10 & 6 & \cdots & 8 & \cdots & 10 \\
6 & \ddots & 8 & & 10 & 10 & 6 & & 8 & & 10 \\
\vdots & & & & & & & & & & \vdots \\
0 & \cdots & 2 & \cdots & 4 & 4 & 0 & \cdots & 2 & \cdots & 4 \\
0 & & 0 & & 2 & 2 & 0 & & 0 & & 2 \\
\vdots & & & & & & & & & \ddots & \vdots \\
0 & \cdots & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & \cdots & 0
\end{pmatrix}
\end{array}
$$

(b) *Education*

Fig. 4: Quasi-identifiers cost matrices [37]

III. Our CGM-based parallel algorithms for k-anonymity

The strong dependencies between data in consecutive iterations limit the available sources of parallelism for this problem. In the following sections, we present three parallel algorithms based on the CGM model to minimize computation time.

Algorithm 1 Greedy k-anonymity algorithm

procedure GkAA(T, k, μ)
  while $\exists C \in \mathcal{C}(T)$ as $\|C\| < k$ **do**
    Choose a class C_{small} of $\mathcal{C}(T)$
    $C \leftarrow$ SEARCH$(\mathcal{C}(T), C_{small}, \mu)$
    $C_{merge} \leftarrow Merge(C_{small}, C)$
    $\mathcal{C}(T) \leftarrow \mathcal{C}(T) \setminus \{C_{small}, C\} \cup Merge(C_{small}, C)$

Algorithm 2 Search of the suitable pair of equivalence classes to merge

function SEARCH$(\mathcal{C}(T), C_{small}, \mu)$
  while $\mathcal{C}(T) \neq \emptyset$ **do**
    Choose a class C from $\mathcal{C}(T)$
    if $\mu(C_{small}, C) < min$ **then**
      $min \leftarrow \mu(C_{small}, C)$
      $C_{min} \leftarrow C$
    $\mathcal{C}(T) \leftarrow \mathcal{C}(T) \setminus \{C\}$
  return C_{min}

A. PGkAA: CGM-based parallel algorithm based on the partitioning of the search space

We introduce the PGkAA algorithm, which aims to reduce the computation cost of the k-anonymity problem by employing parallelism in the search for suitable pairs of equivalence classes to merge. In this approach, we propose a straightforward partitioning method. Indeed, the computation of merging costs for two disjoint pairs of equivalence classes is entirely independent.

The set of equivalence classes, denoted as $\mathcal{C}(T)$, is partitioned into s subsets of equivalence classes during each iteration to facilitate the search. Furthermore, each processor searches within a subset of equivalence classes, denoted as

Algorithm 3 CGM-based parallel algorithm based on the partitioning of the search space

procedure PGkAA(T, k, μ, p)
  Sort in ascending order of size the $C \in \mathcal{C}(T)$
  while $\exists C \in \mathcal{C}(T)$ as $\|C\| < k$ **do**
    Choose a class C from $\mathcal{C}(T)$
    $C \leftarrow$ PSEARCH($\mathcal{C}(T)$, C_{small}, μ, p)
    One-To-All communication of C
    $C_{merge} \leftarrow Merge(C_{small}, C)$
    $\mathcal{C}(T) \leftarrow \mathcal{C}(T) \setminus \{C_{small}, C\} \cup C_{merge}$

Algorithm 4 Search for the suitable pair of equivalence classes to merge in parallel

function PSEARCH($\mathcal{C}(T)$, C_{small}, μ, p)
  Each processor get its subset of r/p equivalence classes $\delta\mathcal{C}(T)$
  while $\delta\mathcal{C}(T) \neq \emptyset$ **do**
    Choose a class δC from $\delta\mathcal{C}(T)$
    if $\mu(C_{small}, \delta C) < min$ **then**
      $min \leftarrow \mu(C_{small}, \delta C)$
      $\delta C_{min} \leftarrow \delta C$
    $\delta\mathcal{C}(T) \leftarrow \delta\mathcal{C}(T) \setminus \{\delta C\}$
  All-to-One communication of δC_{min} to the master processor
  return SEARCH($\delta\mathcal{C}_{min}(T), C_{small}, \mu$)

$\delta\mathcal{C}(T)$, ensuring that this subset contains at least $(r/p + 1)$ equivalence classes, where p is the number of processors.

Algorithm 3 summarizes the main steps of our distributed parallel solution. The parallel search function proceeds the search within a smaller range of equivalence classes that help speed up the global solution. Algorithm 4 summarizes the steps of PSEARCH (in Algorithm 3).

Each processor partitions the set of equivalence classes, denoted as $\mathcal{C}(T)$, and selects its specific subset, $\delta\mathcal{C}(T)$, for local computation based on its rank. Subsequently, each processor conducts a search within its designated subset of equivalence classes. Upon completion of their respective searches, the slave processors transmit the equivalence class with the minimum merge cost to the master processor via an All-to-One communication. The master processor then carries out the merging process and subsequently shares the updated set of equivalences with the other processors using a One-to-All communication.

Theorem 2. *The PGkAA algorithm runs in $\mathcal{O}\left(n^2/p\right)$ execution time with $\mathcal{O}\left(np\right)$ communications rounds.*

It is worth noting that the number of communication rounds in the PGkAA algorithm is heavily dependent on the size of the database, denoted as n. This dependence poses a significant challenge when attempting to scale this solution for use with large databases. In fact, the communication time emerges as a notable drawback in this context.

B. PGPkAA: CGM-based parallel algorithm based on the partitioning of the dataset

The main goal of this approach is to reduce the number of communication rounds generated by the PGkAA algorithm, thereby enhancing the overall solution's speedup. Our strategy involves simplifying the problem by working with a dataset consisting of (n/p) records, which constitutes a partial dataset. Sang et al. [10] demonstrated that partitioning the dataset accelerated their solution, albeit at the expense of introducing a new challenge: information loss, which escalates as the number of partitions increases. Indeed, the dissimilarities observed between partitions represent one of the primary factors contributing to the expansion of information loss during the anonymization process.

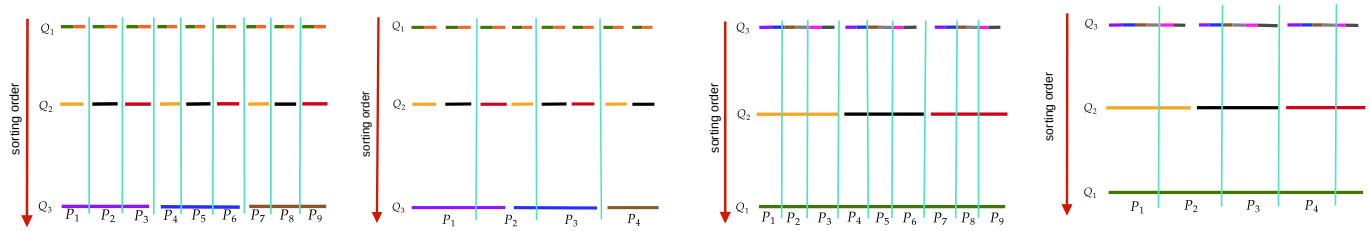
1) Data reorganization: It is important to note that, attributes with higher generalization costs tend to result in greater information loss when the dataset is randomly partitioned with a high number of partitions. We propose the use of data reorganization as a preprocessing technique to reduce information loss of k-anonymization in distributed memory environments. This technique involves arranging records in the database in a way that groups the more similar lines together. By suitably reorganizing the data, we can define the order in which quasi-identifier attributes are aggregated. In this paper, we specifically focus on a multidimensional stable sorting approach based on attributes generalization hierarchy levels. When sorting the data, two schemes can be considered. Scheme 1 (resp. scheme 2) consider sorting the attributes from the highest (resp. lowest) generalization hierarchy level to the lowest (resp. highest). The generalization hierarchy level indicates the degree of generalization for two attribute values, where a higher level implies a lower generalization cost.

Let $\mathcal{Q} = \{Q_1, \ldots, Q_t\}$ be a set of quasi-identifiers. Let $l_1, \ldots, l_t$ be the generalization hierarchy level associated with $Q_1, \ldots, Q_t$ respectively and $n_1, \ldots, n_t$ the number of unique values for each quasi-identifier. In the case of performing horizontal partitioning on a dataset, sorting the dataset based on its quasi-identifier attributes in ascending order of the cost of generalization for each attribute helps to preserve information loss in a distributed memory environment. In this article, we consider that a higher level of generalization hierarchy for an attribute implies a lower cost of generalizing its values [11].

Lemma 1. *For two attributes Q_i, Q_j with $l_i = l_j$ and $i \neq j$, if $n_i < n_j$ the dataset must be sorted according to Q_i before being sorted according to Q_j.*

For example, let us consider a situation where we have a dataset T with three quasi-identifiers Q_1, Q_2, Q_3 associated with generalization hierarchy levels l_1, l_2, l_3 respectively. Let us assume that $l_1 < l_2 < l_3$ (i.e. Q_1 has the highest generalization cost penalty, followed by Q_2, then Q_3). Let Q_1, Q_2, Q_3 be assigned to 2, 3, and 6 distinct values, respectively. Fig. 5 illustrates how the choice of reorganization scheme can impact the resulting information loss after partitioning the dataset. In particular, Fig. 5a and 5c (resp. Fig. 5b and 5d) illustrate the partitioning of T, with each partition having 2 (resp. 4) records. Note that Fig. 5a and 5b (resp. Fig. 5c and 5d) adopt reorganization in scheme 2 (resp. scheme 1) and are crops of Fig. 12a (resp. Fig. 12b) in appendix A. In Fig. 5a and 5b, we start by sorting the whole dataset on Q_1 (step 1), then on Q_2 (step 2), and lastly on Q_3 (step 3), while in Fig. 5c and 5d, the attributes are sorted in the reverse order. Note that, each color in these figures represents unique values for the correspondent attribute and sorting is applied on all lines of the database.

Therefore, Fig. 5a shows that the values of Q_1 will be



(a) 2-partitioning using reorganization in scheme 1 (b) 4-partitioning using reorganization in scheme 1 (c) 2-partitioning using reorganization in scheme 2 (d) 4-partitioning using reorganization in scheme 2

Fig. 5: Example of reorganization influence on a 2-partitioning and 4-partitioning of a dataset

Algorithm 5 CGM-based parallel algorithm based on the dataset partitioning with reorganization

1: **procedure** PGPkAA(T, k, μ, p)
2: Reorganize the dataset T
3: Partition the dataset T into p subsets ΔT
4: GkAA ($\delta T, k, \mu$), where $\delta T \in \Delta T$
5: Slave processors send their local k-anonymous dataset $\delta T_{\mu,k}$ to the master processor for aggregation using All-to-One communication

locally generalized, leading to a significant global information loss. This is due to the high generalization cost associated with Q_1, as it has the lowest hierarchy level. Conversely, Fig. 5c shows that the values of Q_1 will not undergo generalization.

In the other scenario, Fig. 5b depicts how the values of Q_1 and Q_2 will be generalized in all partitions while Q_3 will be generalized in some partitions. This results in increased information loss. On the other hand, Fig. 5d demonstrates that only Q_3 will be generalized in all partitions, thereby preserving the information loss to a greater extent.

Based on the above observations, scheme 2 might be more suitable for preserving information loss compared to scheme 1. In the subsequent parallel algorithms, we adopt the approach of reorganizing the dataset following scheme 2.

It is important to acknowledge that the partitioning process may potentially introduce some disruption to the obtained results, with a slight increase in terms of information loss [10], [31] compared to the centralized GkAA algorithm [12]. However, we anticipate that the time savings achieved through our method will outweigh this trade-off.

Overall, we propose the PGPkAA algorithm to strike a balance between minimizing computation time and controlling information loss, with the expectation that the advantages of our approach will outweigh any potential drawbacks.

2) Partial greedy parallel k-anonymity algorithm: Each processor reorganizes the dataset, dividing it into p subsets. It selects a subset corresponding to its rank and independently applies the centralized GkAA algorithm [12] to its respective partition. After completing the local computation round, each processor transmits its locally generated k-anonymous dataset to the master processor (or coordinator) for aggregation. Algorithm 5 summarizes the main steps of our solution.

Theorem 3. *The PGPkAA algorithm runs in $\mathcal{O}\left(n^2/p^2\right)$ execution time with $\mathcal{O}\left(p\right)$ with communication rounds.*

Algorithm 6 Hybridization of PGkAA and PGPkAA

1: **procedure** H-PGPkAA(T, k, μ, p)
2: Determine the number of subsets $b = f(p) = \lfloor \log_2 p \rfloor + 1$
3: Reorganize the dataset T
4: Partition T into b subsets ΔT
5: Form blocks of processors ΔB, each consisting of $\left(\lfloor \frac{p}{b} \rfloor + 1\right)$
6: PGkAA(δT, k, μ, δB), where $\delta B \in \Delta B$ and $\delta T \in \Delta T$
7: Master processors of each block send their local k-anonymous tables $\delta T_{\mu,k}$ to the main master processor for aggregation using All-to-One communication

C. H-PGPkAA: CGM-based parallel algorithm based on the partitioning of the search space and the dataset

The number of dataset partitions, or fragments, is determined by the number of processors. As p increases, the subsets become smaller, resulting in an increased level of information loss in the resulting k-anonymous tables. While the PGPkAA solution achieved significant speedup, it also led to a slight increase in information loss, particularly for higher values of p and smaller datasets. To address this issue, we propose the H-PGPkAA algorithm, which combines the strengths of both the PGkAA and PGPkAA algorithms to further minimize information loss while achieving k-anonymity.

To achieve this objective, we reduce the number of subsets by partitioning the dataset using a given positive definite function. We define a function $b = f(p) = (\lfloor \log_2 p \rfloor + 1)$, which determines the number of subsets required for the PGPkAA parallel algorithm. This function assists us in determining the minimum number of processors needed to minimize the number of communication rounds. Consequently, for a given number of processors p, we leverage these remaining $(p - b)$ processors to expedite the search process within the PGkAA algorithm (see Algorithm 3) by forming dedicated processor blocks. Each block comprises (p/b) processors working on the same subset, each processing a maximum of $\left(\frac{n}{b} + 1\right)$ records.

Processors within the same block collaborate to search for the appropriate pair of equivalence classes for merging. Following the completion of the search within each block, the slave processors communicate their search results to the master processor of the block (typically the processor with the lowest rank) using an All-to-One communication. The master processor performs the merge operation and returns the updated set of equivalence classes to the slave processors through a One-to-All communication. Once each block finishes its execution, the master processor of each block utilizes an All-to-One

communication to transmit its local k-anonymous tables to the main master processor responsible for aggregation. This processor serves as the master of the first block.

Algorithm 6 offers an overview of our solution, outlining the key steps involved in achieving k-anonymity while minimizing information loss through the use of data partitioning, block-based processing, and communication among processors.

Theorem 4. *The H-PGPkAA algorithm runs in $\mathcal{O}\left(\frac{n^2}{p\log_2 p}\right)$ execution time with $\mathcal{O}\left(\frac{np}{(\log_2 p)^2}\right)$ communication rounds.*

IV. Results

We conducted a comparison between our distributed parallel algorithms and the centralized GkAA algorithm [11], [12]. The implementations were done in C++ and Python3, and they were executed on a CentOS Linux operating system within the MatriCS platform at the University of Picardie Jules Verne [39]. To facilitate communication between processors, we utilized the MPI (Message Parsing Interface) library and mpi4py framework for Python. Our experimental evaluations were performed using two datasets: the Adult dataset sourced from the University of California in Irvine's machine learning repository [34] and the Florida voters list dataset [40]. The Adult dataset is a commonly used benchmark for experimenting with k-anonymity algorithms.

The reported results are based on various combinations of the triplet (n, p, k). Here, n represents the dataset size, with values selected from $\{30{,}162\}$ for the Adult dataset and $\{60{,}000;\ 120{,}000;\ 240{,}000;\ 500{,}000;\ 1{,}000{,}000\}$ for the Florida dataset. The variable p denotes the number of processors, chosen from $\{1, 2, 4, 8, 16, 32, 64, 128\}$, and k the k-anonymity constraint , with values taken from $\{2, 3, 5, 10, 100\}$. In cases where p is set to 1, the GkAA algorithm [12] is executed.

The parallel speedup metric was employed to assess the relative speed improvement achieved by our parallel algorithms in comparison to the sequential algorithm. For a deeper understanding, Kruskal et al. discussed this topic in [41].

A. Adult dataset evaluation

For the evaluation of the Adult dataset, we carefully selected nine quasi-identifier attributes: *Age, Gender, Race, Marital status, Education, Native country, Work class, Occupation, and Salary*. Any tuples with missing values were meticulously removed, resulting in a dataset containing precisely 30,162 records. These records were subsequently organized into 19,502 pre-existing equivalence classes. Table I furnishes a succinct summary description of the Adult dataset.

The anonymization process for this dataset was initiated to generate a k-anonymous dataset, with a primary focus on optimizing a metric that minimizes information loss. The anonymization procedure closely follows the methodology outlined in [11]. Initially, we applied the GkAA algorithm, followed by the utilization of three parallel algorithms introduced in this paper: PGkAA, PGPkAA, and H-PGPkAA.

Table V shows an overview of the execution times for the algorithms discussed in this paper, specifically concerning the Adult dataset. These results serve as crucial inputs to compute the speedup and efficiency of the parallel algorithms.

TABLE I: Description of Adult dataset attributes

Attributes	Number of values	Generalization (graph's height)
Age	74	5−, 10−, 20−year intervals (5)
Gender	2	Suppression (2)
Race	7	Suppression (2)
Marital status	5	Hierarchy (3)
Education	16	Hierarchy (4)
Native country	41	Hierarchy (3)
Work class	7	Hierarchy (3)
Occupation	14	Hierarchy (3)
Salary	2	Suppression (2)

1) Evaluation of the overall computation time: Fig. 6a demonstrates that the execution time increases as k increases. This observation aligns with the expected behavior, as higher values of k naturally lead to longer processing times for generating the anonymous database.

Fig. 6b shows that the overall execution time decreases as the number of processors increases. This effect is particularly pronounced in the PGPkAA approach compared to the PGkAA approach. The PGPkAA algorithm effectively reduces execution time by optimizing processor utilization. Furthermore, with the increasing number of processors, each processor handles smaller subsets of the k-anonymity problem, resulting in improved performance. This acceleration can be attributed to the quadratic complexity of the sequential algorithm concerning the number of records in the database. By partitioning the database into multiple subsets, we achieve a theoretical speedup proportional to the square of the number of processors.

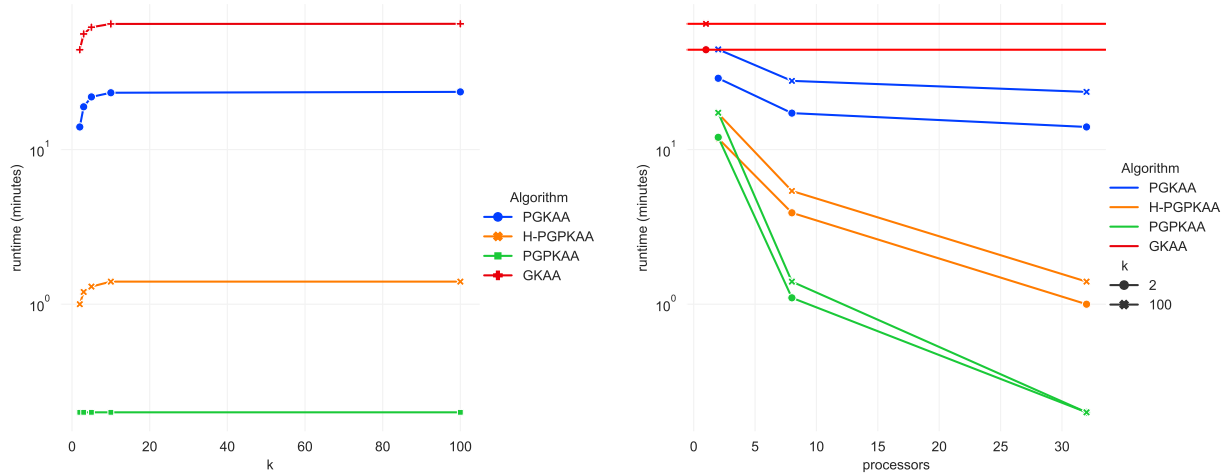
However, it is important to note that excessive parallelism may come at the cost of compromising data integrity. In Section IV-A3, we will delve into the impact of parallelism on information loss within the k-anonymous database.

Table II in Appendix C presents evidence that the PGkAA algorithm, on average, exhibits a speedup of $2.5\times$ compared to the sequential algorithm when executed on 32 processors. Likewise, the PGPkAA algorithm showcases a superlinear speedup, being $300\times$ faster than the GkAA algorithm. These results underscore the scalability of the PGPkAA approach, particularly in scenarios involving larger datasets.

2) Evaluation of the overall communication time: Fig. 7 illustrates varying behaviors in communication time as the number of processors increases for the three algorithms. Similar trends were also observed in the results reported in [10]. For the PGPkAA algorithm, communication time decreases as the number of processors increases. This suggests that the algorithm benefits from parallelization, as more processors enable faster inter-processor communication.

Conversely, the communication time increases for the PGkAA algorithm with an increase in the number of processors. This is primarily due to the fact that, in each iteration of the loop, two rounds of communications are performed, resulting in a greater communication overhead as the number of processors grows $(2 \times p \times n)$.

Regarding the H-PGPkAA algorithm, communication time



(a) $p \in \{1, 32\}$ and $k \in \{2, 3, 5, 10, 100\}$

(b) $p \in \{1, 2, 8, 32\}$ and $k \in \{2, 100\}$

Fig. 6: Overall execution time with $n = 30162$, $p \in \{1, 2, 8, 32\}$, and $k \in \{2, 3, 5, 10, 100\}$ for Adult dataset

TABLE II: Speedup of parallel algorithms with $n = 30162$, $p \in \{2, 8, 32\}$, and $k \in \{2, 3, 5, 10, 100\}$ for Adult dataset

Algorithm	k = 2			k = 3			k = 5			k = 10			k = 100		
	Number of processors			Number of processors			Number of processors			Number of processors			Number of processors		
	2	8	32	2	8	32	2	8	32	2	8	32	2	8	32
PGkAA	1.53	2.58	3.15	1.51	2.47	2.96	1.5	2.33	2.82	1.48	2.29	2.78	1.46	2.34	2.76
PGPkAA	3.68	42.07	278.99	3.69	41.3	294.2	3.76	42.71	308.97	3.77	43.7	327.18	3.75	45.67	309.56
H-PGPkAA	3.73	11.36	46.01	3.77	11.65	46.13	3.81	11.87	48.21	3.8	11.29	46.8	3.8	12.07	47.74

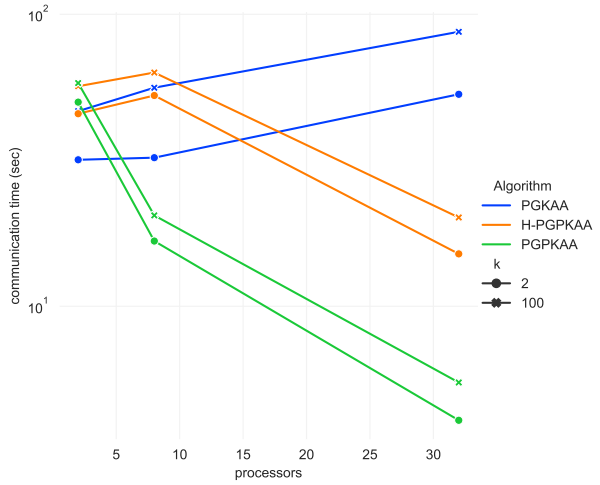


Fig. 7: Overall communication time with n = 30,162, $p \in \{2, 8, 32\}$, and $k \in \{2, 100\}$ for Adult dataset

decreases as the number of blocks increases. This reduction in communication time is a direct result of using blocks, which helps minimize the number of processors involved in the search for the best pair of equivalence classes to merge during each iteration. The overall communication time decreases by reducing the number of processors engaged in this task.

In summary, these findings underscore the critical importance of carefully considering communication overhead and optimizing communication strategies during the design and implementation of parallel algorithms. As the number of processors increases, the amount of data to be transmitted decreases. This reduction occurs because the workload is distributed across more processors, resulting in smaller subsets for each processor to manage. Consequently, communication time between processors decreases.

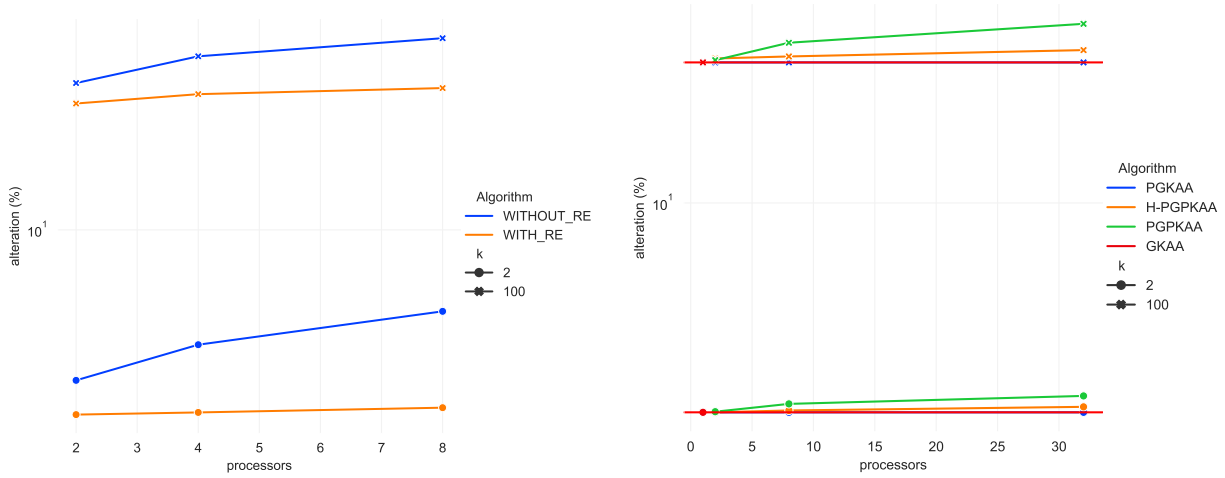
As an illustration, for $p = 32$ and $k = 100$, the PGPkAA algorithm showcases a communication time of 5 seconds, whereas the PGkAA algorithm records a communication time of 82 seconds. Thus, PGPkAA demonstrates significantly faster communication compared to PGkAA. In Section IV-B, our experiments will solely focus on the PGPkAA and H-PGPkAA algorithms. The decision to exclude the PGkAA algorithm is rooted in its escalating communication time as the number of processors and dataset size increase. The inefficiency observed in the PGkAA algorithm is primarily attributed to its heavy dependence on the database size n.

3) Evaluation of the information loss: For the PGPkAA and H-PGPkAA algorithms, the subsequent results are employed to compute the distributed information loss:

$$A_{\mu}^{par}(T) = \sum_{\delta T \in \Delta T} A_{\mu}(\delta T) \times \frac{\|\delta T\|}{n} \tag{2}$$

where ΔT is the set of k-anonymous tables produced by processors or blocks of processors, $\|\delta T\|$ signifies the size of the subset, n is the size of the original dataset, and μ stands for a specific optimization metric. The following sections evaluate and compare the amount of information loss when executing the PGPkAA algorithm with and without data reorganization.

Fig. 8a illustrates that the difference in information loss increases as the value of k rises between the approach with



(a) PGPkAA algorithm with and without reorganization

(b) $p \in \{1, 2, 8, 32\}$, and $k \in \{2, 100\}$

Fig. 8: Information loss with n = 30,162, $p \in \{1, 2, 8, 32\}$, and $k \in \{2, 100\}$ for Adult dataset

and without data reorganization. This contrast becomes more pronounced with a larger number of processors. However, despite the reduction in computation time, the algorithm remains impractical for real-life applications. For example, there is an observed information loss of 46% for $p = 8$ and $k = 100$ when the data reorganization is not employed. In contrast, the information loss is reduced to 31% when reorganization is utilized. In the subsequent analysis, the PGPkAA and H-PGPkAA algorithms were executed with data reorganization as a preprocessing step to enhance information loss.

Fig. 8b demonstrates that both the sequential and PGkAA algorithms yield the same level of information preservation. However, as mentioned before, the PGPkAA approach tends to slightly increase information loss compared to the GkAA algorithm. This difference becomes more pronounced when using higher values of p and k. Specifically, as the number of processors increases, the size of the local dataset on each processor decreases. This reduction in dataset size diminishes the search space for identifying the most appropriate equivalence classes to merge, potentially resulting in higher information loss. It was observed that, for smaller values of k, the information loss incurred by the PGPkAA approach is relatively close to that of the sequential approach. As an example, Table VIII in Appendix C presents the information loss for the PGPkAA algorithm for $p = 32$ and $k = 2$, the information loss is 2.56%, while the information loss of the GkAA algorithm is 2.28% for $k = 2$.

Certainly, the reorganization preprocessing conducted on the data enables the PGPkAA approach to distribute computations across multiple processors while maintaining reasonable information loss in the resulting k-anonymous database. This preprocessing step plays a crucial role in optimizing the distribution of workload and minimizing the overall impact on data integrity. As a result, the PGPkAA approach can effectively leverage parallel processing capabilities without compromising the quality and privacy guarantees of the final k-anonymous database.

TABLE III: Description of Florida dataset attributes

Attributes	Number of values	Generalization (graph's height)
Residence Zipcode	1026	500−, 1000−zipcode intervals (4)
Gender	2	Suppression (2)
Race	8	Suppression (2)
Year of birth	109	10−, 20−, 30−year intervals (5)
Party affiliation	10	Suppression (2)

B. Florida dataset evaluation

The Florida dataset used in this study is a publicly available dataset accessible online. It is derived from the Florida Voter Registration System and contains information about registered voters as of the previous month's end. The raw dataset initially consists of 38 attributes and includes a total of 14 million records. However, for these experiments, only 5 specific attributes were selected, resulting in a subset of one million records. These records were organized into 151,061 pre-existing equivalence classes. Table III provides a summary of the dataset attributes considered within the scope of this research. Table VI and Table VII in Appendix C provide a summary of the overall execution time for the algorithms discussed in this paper.

1) Evaluation of the overall computation and communication time: As expected, fig. 9a showcase how the overall computation time increases with the growth of k. This behavior is typical as the creation of large equivalence classes results in increased computational demands. Similar to what was discussed in section IV-A1, Fig. 9b demonstrates the reduction in execution time for the PGPkAA parallel algorithm as the number of processors increases. This trend becomes more prominent when dealing with larger databases. Additionally, Fig. 9c and 9d showcase the decrease in overall execution time as the number of processors increases, indicating the scalability of the PGPkAA approach when handling larger datasets. For example, according to Table IV, on 128 processors, the PGPkAA algorithm achieves an average speedup of $770\times$ faster than the centralized GkAA algorithm (about

600% computation time saving).

Fig. 10a and 10b show that the communication time decreases as the number of processors increases. This observation is attributed to the characteristics of the CGM model, where the communication time is heavily influenced by the size of the data transmitted through the network and the number of data items to be transmitted. As a result, when the size of the data in the network is smaller, the transmission speed increases, leading to a reduction in communication time.

2) Evaluation of the information loss: Fig. 11a and 11b reveal that the information loss decreases as the size of the database increases for both the sequential and parallel solutions. This indicates that our parallel solutions can effectively handle larger datasets while maintaining lower information loss.

Zooming in on the case of n = 1,000,000, Fig. 11c and 11d highlight the minimal increase in terms of information loss for both the PGPkAA and H-PGPkAA algorithms compared to the GkAA algorithm. This further emphasizes the efficiency of the PGPkAA approach in managing large-scale datasets. For example, according to Table IX, the PGPkAA algorithm achieves an information loss of 0.91% with $p = 128$ and $k = 10$, while the information loss of the GkAA algorithm is 0.63%. These results demonstrate the effectiveness of the parallel approach in handling large-scale datasets with minimal information loss.

V. Conclusion

In this paper, we first proposed data reorganization as a preprocessing step to enhance information loss while achieving anonymity for large-scale databases in a distributed memory environment. The preprocessing has proven to be effective. It has significantly reduced information loss in a distributed memory environment, especially with the increasing number of processors and the constraint k. As baseline, we proposed the PGkAA parallel algorithm based on a straightforward partitioning approach. This solution has proven to be more suitable for small-sized datasets and has the advantage of achieving the same information loss as the GkAA algorithm while speeding up the process by an average factor of 2. Furthermore, we proposed two avanced parallel algorithms (PGPkAA and H-PGPkAA) based on the CGM model, which uses data reorganization as a preprocessing step. Through experimental evaluation, we observed that the substantial speedup achieved by the PGPkAA algorithm (600% computation time saving) compensates for the slight increase in information loss compared to the GkAA algorithm. Notably, the PGPkAA algorithm exhibited scalability and efficiency by effectively handling increasing values of n and k as the number of processors increased. Therefore, this solution proves to be both efficient and scalable for databases. The H-PGPkAA algorithm showed good acceleration with slightly equal information loss as that of the GkAA algorithm. Additionally, we believe that the data reorganization used in these algorithms can be applied to improve data utility while achieving k-anonymity in other distributed parallel algorithms employing horizontal data partitioning, including the GCCG algorithm [10] or the extended Mondrian algorithm [31].

As future directions for this work, we aim to explore methods that can enhance the data utility of the PGPkAA parallel algorithm. This can be achieved by reducing the number of subsets in the dataset. Additionally, we plan to leverage artificial intelligence techniques, such as machine learning approaches (similar to those used by Viton et al. [35]), to further improve data utility. Furthermore, we intend to enhance the speedup of our parallel solutions by incorporating automatic parallelism in shared memory architectures. This can be achieved through the utilization of GPUs or FPGAs, which have the potential to significantly accelerate algorithm execution. By pursuing these avenues, we aim to not only improve the data utility of the parallel algorithms but also enhance their performance and efficiency on modern shared memory architectures.

References

[1] T. Carvalho, N. Moniz, P. Faria, and L. Antunes, "Survey on privacy-preserving techniques for microdata publication," *ACM Comput. Surv.*, vol. 55, no. 14s, jul 2023.

[2] C. Li, H. Shirani-Mehr, and X. Yang, "Protecting individual information against inference attacks in data publishing," in *Advances in Databases: Concepts, Systems and Applications*, R. Kotagiri, P. R. Krishna, M. Mohania, and E. Nantajeewarawat, Eds. Springer Berlin Heidelberg, 2007, vol. 4443, pp. 422–433, series Title: Lecture Notes in Computer Science.

[3] P. Samarati and L. Sweeney, "Generalizing data to provide anonymity when disclosing information," *PODS*, vol. 98, p. 188, 1998.

[4] P. Samarati, "Protecting respondents identities in microdata release," *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, no. 6, pp. 1010–1027, 11 2001.

[5] L. Sweeney, "Achieving k-anonymity privacy protection using generalization and suppression," *International Journal of Uncertainty, Puzziness and Knowledge-Based Systems*, vol. 10, no. 5, pp. 571–588, 2002.

[6] A. Meyerson and R. Williams, "On the complexity of optimal k-anonymity," *In Proceedings of the Twenty-third ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, vol. 4, pp. 223–228, 2004.

[7] J. Li, R. C.-W. Wong, A. W.-C. Fu, and J. Pei, "Achieving k-anonymity by clustering in attribute hierarchical structures," *Data Warehousing and Knowledge Discovery*, pp. 405–416, 2006.

[8] J. Xu, W. Wang, J. Pei, X. Wang, B. Shi, and A. W.-C. Fu, "Utility-based anonymization using local recoding," *In Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD*, pp. 785–79, 2006.

[9] J.-W. Byun, A. Kamra, E. Bertino, and N. Li, "Efficient k-anonymization using clustering techniques," *E., eds., Advances in Databases: Concepts, Systems and Applications*, pp. 188–200, 2007.

[10] N. Sang, X. Mengbo, and Q. Quan, "Clustering based k-anonymity algorithm for privacy preservation," *International Journal of Network Security*, vol. 19, no. 6, pp. 1062–1071, 11 2017.

[11] C. Mauger, G. Le Mahec, and G. Dequen, "Modeling and evaluation of k-anonymization metrics," *Privacy Preserving Artificial Intelligence Workshop of AAAI20*, pp. 1–8, 2020.

[12] ——, "Optimizing privacy and data utility: Metrics and strategies," *Transactions on Data Privacy*, vol. 16, no. 3, pp. 153–189, 2023.

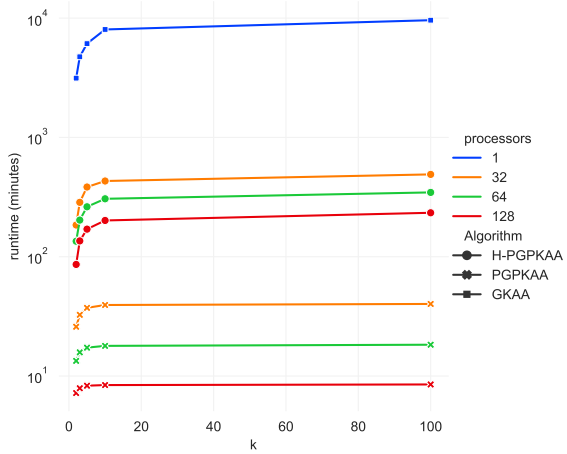
[13] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, p. 107–113, jan 2008.

[14] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken, "Logp: Towards a realistic model of parallel computation," *SIGPLAN Not.*, vol. 28, no. 7, p. 1–12, jul 1993. [Online]. Available: https://doi.org/10.1145/173284.155333

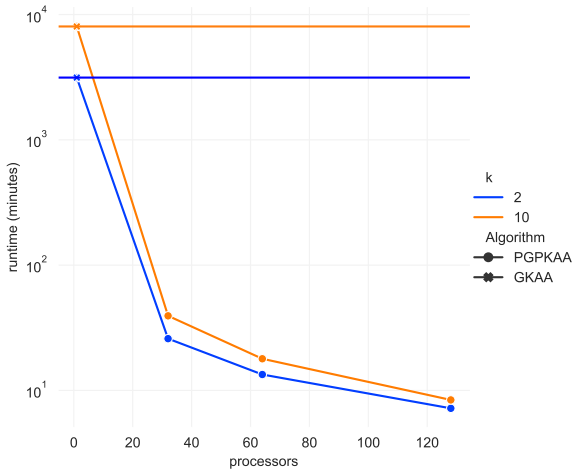
[15] L. G. Valiant, S. Skyum, S. Berkowitz, and C. Rackoff, "Fast Parallel Computation of Polynomials Using Few Processors," *SIAM Journal on Computing*, vol. 12, no. 4, pp. 641–644, 1983.

[16] F. Dehne, A. Fabri, and A. Rau-Chaplin, "Scalable parallel geometric algorithms for coarse grained multicomputers," *In Proceedings of the ninth annual symposium on Computational geometry*, 1993.

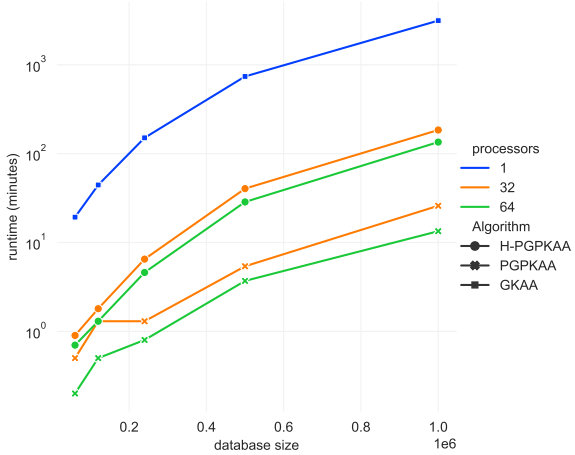
[17] Y. Feng and L. Wang, "Distributed ItemCF Recommendation Algorithm Based on the Combination of MapReduce and Hive," *Electronics*, vol. 12, no. 16, 2023.



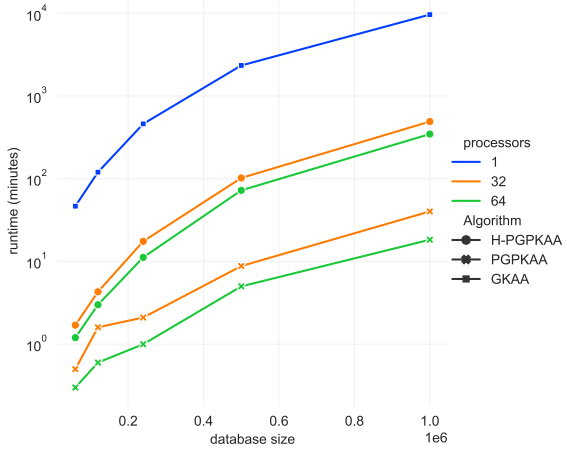
(a) $n = 10^6$, $p \in \{1, 32, 64, 128\}$, and $k \in \{2, 3, 5, 10, 100\}$



(b) $n = 10^6$, $p \in \{1, 32, 64, 128\}$, and $k \in \{2, 10\}$

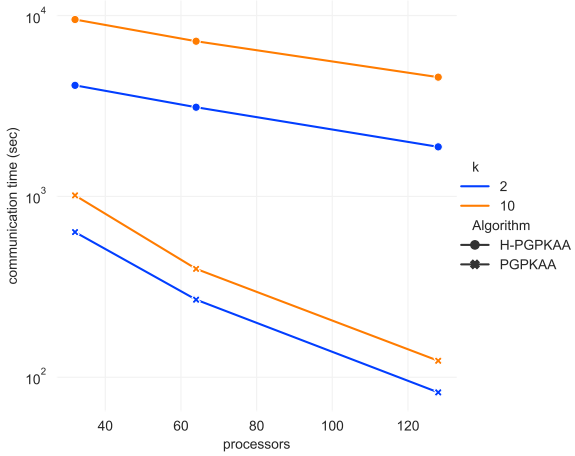


(c) $n \in \{6 \times 10^4, \ldots, 10^6\}$, $p \in \{1, 32, 64\}$, and $k = 2$

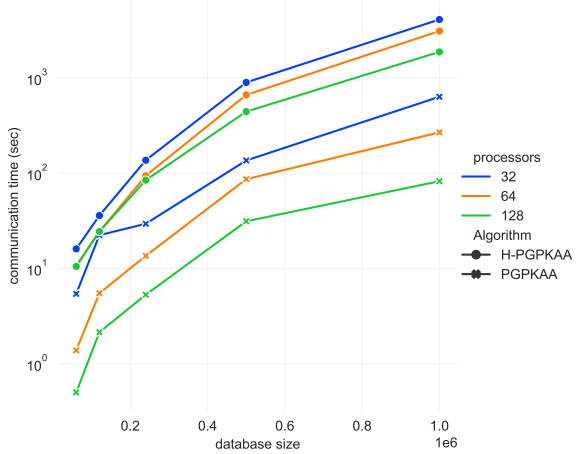


(d) $n \in \{6 \times 10^4, \ldots, 10^6\}$, $p \in \{1, 32, 64\}$, and $k = 100$

Fig. 9: Overall execution time with $n \in \{6 \times 10^4, \ldots, 10^6\}$, $p \in \{1, 32, 64, 128\}$, and $k \in \{2, 3, 5, 10, 100\}$ for Florida dataset



(a) $n = 10^6$, $p \in \{32, 64, 128\}$, and $k \in \{2, 10\}$

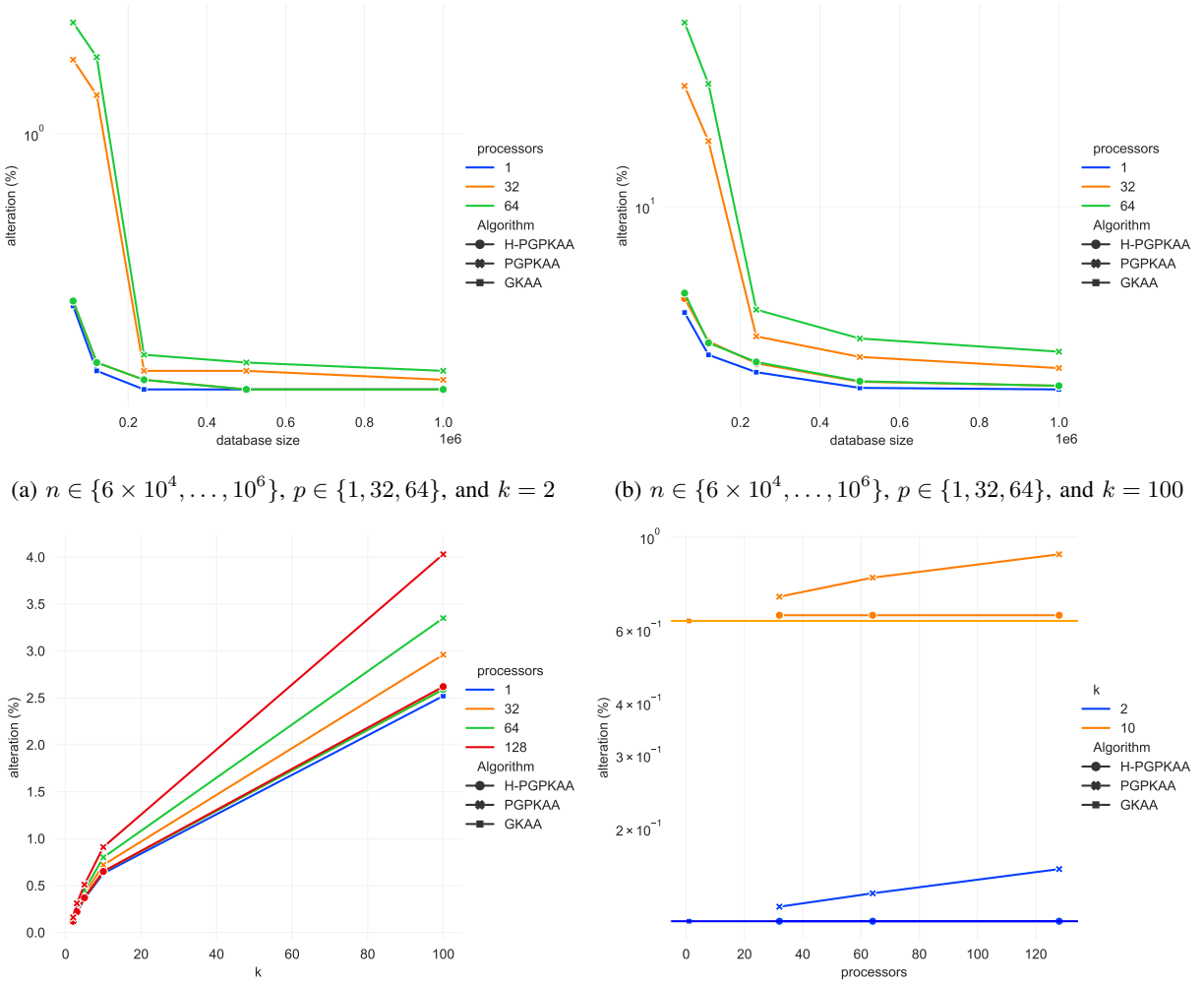


(b) $n \in \{6 \times 10^4, \ldots, 10^6\}$, $p \in \{32, 64, 128\}$, and $k = 2$

Fig. 10: Overall communication time with $n \in \{6 \times 10^4, \ldots, 10^6\}$, $p \in \{32, 64, 128\}$, and $k \in \{2, 10\}$ for Florida dataset

TABLE IV: Speedup of parallel algorithms with $n = 10^6$, $p \in \{32, 64, 128\}$, and $k \in \{2, 3, 5, 10, 100\}$ for Florida dataset

Algorithm	k = 2			k = 3			k = 5			k = 10			k = 100		
	Number of processors			Number of processors			Number of processors			Number of processors			Number of processors		
	32	64	128	32	64	128	32	64	128	32	64	128	32	64	128
PGPkAA	121.12	234.12	438.3	145.45	300.56	603.42	163.75	354.5	740.64	204.11	448.3	951.08	239.08	524.6	1127.59
H-PGPkAA	17.03	23.29	36.44	16.6	23.39	34.96	15.92	23.32	35.92	18.63	26.26	39.9	19.61	27.77	41.13



(a) $n \in \{6 \times 10^4, \ldots, 10^6\}$, $p \in \{1, 32, 64\}$, and $k = 2$

(b) $n \in \{6 \times 10^4, \ldots, 10^6\}$, $p \in \{1, 32, 64\}$, and $k = 100$

(c) $n = 10^6$, $p \in \{1, 32, 64, 128\}$, and $k \in \{2, 3, 5, 10, 100\}$

(d) $n = 10^6$, $p \in \{1, 32, 64, 128\}$, and $k \in \{2, 10\}$

Fig. 11: Information loss with $n \in \{6 \times 10^4, \ldots, 10^6\}$, $p \in \{1, 32, 64, 128\}$, and $k \in \{2, 3, 5, 10, 100\}$ for Florida dataset

[18] J. L. Zeutouo, T. V. Kengne, and J. F. Myoupo, "High-Performance CGM-Based Parallel Algorithms for Minimum Cost Parenthesizing Problem," *The Journal of Supercomputing*, vol. 78, no. 4, pp. 5306–5332, 2022.

[19] J. L. Zeutouo, V. K. Tchendji, and J. F. Myoupo, "Four-splitting based coarse-grained multicomputer parallel algorithm for the optimal binary search tree problem," *International Journal of Parallel, Emergent and Distributed Systems*, vol. 37, no. 6, pp. 659–679, 2022.

[20] B. Li, Y. Liu, X. Han, and J. Zhang, "Cross-bucket generalization for information and privacy preservation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 3, pp. 449–459, 2018.

[21] B. Li and K. He, "Local generalization and bucketization technique for personalized privacy preservation," *Journal of King Saud University - Computer and Information Sciences*, vol. 35, no. 1, pp. 393–404, 2023.

[22] L. Sweeney, "k-anonymity: A model for protecting privacy," *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, vol. 10, no. 5, pp. 557–5570, 2002.

[23] K. LeFevre, D. DeWitt, and R. Ramakrishnan, "Mondrian multidimensional k-anonymity," in *22nd International Conference on Data Engineering (ICDE'06)*, 2006, pp. 25–25.

[24] V. S. Iyengar, "Transforming data to satisfy privacy constraints," *in Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'02)*, pp. 279–150, 7 2002.

[25] J.-L. Lin and M.-C. Wei, "An efficient clustering method for k-anonymization," in *Proceedings of the 2008 International Workshop on Privacy and Anonymity in Information Society*, ser. PAIS '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 46–50.

[26] J. Luo, X. Yi, F. Han, X. Yang, and X. Yang, "An efficient clustering-based privacy-preserving recommender system," in *Network and System Security*, X. Yuan, G. Bai, C. Alcaraz, and S. Majumdar, Eds. Cham: Springer Nature Switzerland, 2022, pp. 387–405.

[27] J. Wei and C. Chris, "A secure distributed framework for achieving k-anonymity," *The VLDB Journal*, vol. 15, no. 4, pp. 316–333, Aug. 2006.

[28] P. Jurczyk and L. Xiong, "Distributed anonymization: Achieving privacy for both data subjects and data providers," in *Data and Applications Security XXIII*, E. Gudes and J. Vaidya, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 191–207.

[29] N. Mohammed, B. C. M. Fung, P. C. K. Hung, and C.-K. Lee, "Cen-

tralized and distributed anonymization for high-dimensional healthcare data," *ACM Trans. Knowl. Discov. Data*, vol. 4, no. 4, oct 2010.
[30] K. Jong Wook, "Efficiently supporting online privacy-preserving data publishing in a distributed computing environment," *Applied Science*, vol. 11, no. 22, p. 10740, Nov. 2021.
[31] S. De Capitani di Vimercati, D. Facchinetti, S. Foresti, G. Livraga, G. Oldani, S. Paraboschi, M. Rossi, and P. Samarati, "Scalable distributed data anonymization for large datasets," *IEEE Transactions on Big Data*, vol. 9, no. 3, pp. 818–831, 2023.
[32] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan, "Mondrian multidimensional k-anonymity," *In 22nd International Conference on Data Engineering (ICDE'06)*, p. 25, 2006.
[33] L. Sweeney, "Computational disclosure control: A primer on data privacy protection," Ph.D. dissertation, Massachusetts Institute of Technology, 2001.
[34] UCIrvine, "Machine Learning Repository," https://archive.ics.uci.edu/dataset/2/adult, [Online; accessed 13 Sep. 2023].
[35] F. Viton, C. Mauger, G. Dequen, J.-L. Guérin, and G. Le Mahec, "Proportional representation to increase data utility in k-anonymous tables," in *26th IEEE Symposium on Computers and Communications*, Athènes, Greece, Sep. 2021.
[36] M. A. Bender, M. Farach-Colton, G. Pemmasani, S. Skiena, and P. Sumazin, "Lowest common ancestors in trees and directed acyclic graphs," *Journal of Algorithms*, vol. 14, no. 2, pp. 75–94, 2005.
[37] C. Mauger, "Optimisation de l'utilité des données lors d'un processus de k-anonymisation," Ph.D. dissertation, Amiens, 2021.
[38] C. Mauger, G. Le Mahec, and G. Dequen, "Multi-criteria optimization using l-diversity and t-closeness for k-anonymization," *International Workshop on Data Privacy Management International Workshop on Cryptocurrencies and Blockchain Technology*, pp. 73–88, 12 2020.
[39] MatriCS, "Plateforme MatriCS," https://www.matrics.u-picardie.fr/, [Online; accessed 13 Sep. 2023].
[40] "Voter Extract Disk Request - Division of Elections - Florida Department of State," https://dos.myflorida.com/elections/data-statistics/voter-registration-statistics/voter-extract-disk-request, [Online; accessed 13 Sep. 2023].
[41] C. P. Kruskal, L. Rudolph, and M. Snir, "A complexity theory of efficient parallel algorithms," *Theoretical Computer Science*, vol. 71, no. 1, pp. 95–132, 1990.
[42] N. Auger, V. Jugé, C. Nicaud, and C. Pivoteau, "On the worst-case complexity of timsort," *arXiv preprint arXiv:1805.08612*, 2018.

PLACE PHOTO HERE

Arsème Vadèle Djeufack Nanfack received a Master's degree in Networks and Distributed Services from the University of Dschang, Cameroon in 2021. He is pursuing his Ph.D. at the MIS laboratory of the University of Picardie Jules Verne, France. His work focuses on privacy preserving, data anonymization, distributed computing, database partitioning and data aggregation.

PLACE PHOTO HERE

Gaël Le Mahec is associate professor at the University of Picardie Jules Verne, member of the MIS laboratory, since September 2010. During his thesis carried out at the Blaise Pascal University of Clermont-Ferrand he studied management and analysis of biological data on Grid-Computing platforms. He is the designer and main contributor of the data manager of the DIET middleware (DAGDA) developed at the Laboratoire de l'Informatique du Parallélisme of the École Normale Supérieure de Lyon, first as a doctoral student then as a research engineer at INRIA and within the start-up SysFera. His work then focused on distributed file systems and he simultaneously worked on the design of a patented security protocol (CryptonAuth) in collaboration with Gilles Dequen, professor at the MIS laboratory. His current main research theme is towards the protection of personal data.

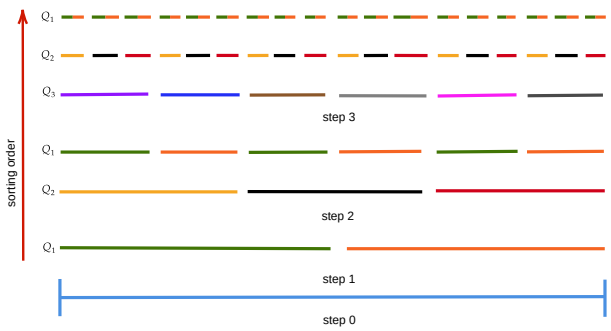
Prof. Gilles Dequen is a French researcher from the MIS Lab, Université de Picardie Jules Verne (UPJV). He received the PhD degree in Computer Science in 2001 and subsequently the HDR degree in 2011. He has been CNRS (French National Scientific Research Center) delegate at the CNRS-LIP6 Laboratory from 2009 to 2010. Gilles Dequen is a Full Professor at UPJV since 2013, and leads the MIS Lab since 2017. Prof. Dequen teaches at UFR des Sciences of UPJV within the Computer Science department in Bachelor and Msc degrees. Mainly, his research interests are centered on Numerical Modeling on one hand, and Decision and Optimization problem solving on the other hand. He applies his expertise on modeling and solving to applications in cybersecurity and healthcare. He is also a founder member of the GRECO Institute dedicated to robotic surgery. He is a PC member of several AI Conferences such as IJCAI-ECAI, AAAI, ICTAI, SAT, etc. and a designated expert supporting several French Government research programs.

Vianney Kengne Tchendji is currently an Associate Professor of computer science at the University of Dschang, Dschang, Cameroon. He received the Ph.D. degree in computer science from the University of Picardie Jules Verne, Amiens, France, in June 2014. Prof. Kengne Tchendji has served as member of program committee of international conferences, and reviewer for many international journals in computer science. His current research interests include parallel algorithms and architectures, network virtualization, internet of things, ad hoc networking and network security.

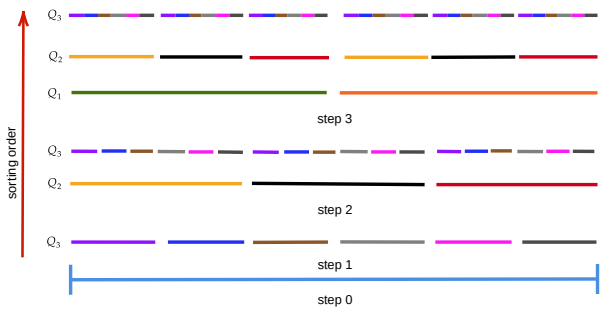
Jerry Lacmou Zeutouo is a researcher at ENS Lyon in Lyon, France. His research interests include parallel computing, partitioning data, and component models. He received his Ph.D. degree in computer science from the University of Dschang, Dschang, Cameroon in December 2022.

Appendix

Appendix A



(a) Lower generalization level to highest generalization level



(b) Highest generalization level to lower generalization level

Fig. 12: Sorting schemes

Appendix B

Proof of Theorem 1

Proof: The number of steps for finding the minimum merge cost class C at each iteration is $(r-1)$ (algorithm 2). At each iteration of the while loop, one equivalence class is extracted from $\mathcal{C}(T)$. Let $t(n)$ be the execution time of the algorithm :

$$t(n) = (r-1) + (r-2) + \cdots + (r-s) = s(2r-s-1)/2$$

with s the number of iterations for the while loop. The following configuration reflects the worst of cases:

- all tuples of T are all distinct according to their quasi-identifiers. That means each tuple forms its own equivalence class, i.e. $r = n$;
- the k-anonymous table has the maximum generalizations. It's a table in which all tuples are identical, i.e. $k = n$ and $s \leq n$.

For these reasons, $t(n) = n(n-1)/2$. Thus, GkAA runs in $O(n^2)$ execution time.

Proof of Theorem 2

Proof: Each processor performs $\frac{(r-1)}{p}$ steps for the search for the minimum merge cost class δC at each iteration, with r as the number of equivalence classes. Thus the computation time for this processor is given by:

$$t_{cals}(n) = \left(\frac{r-1}{p}\right) + \left(\frac{r-2}{p}\right) + \cdots + \left(\frac{r-s}{p}\right) = s\frac{2r-s-1}{2p}$$

With $r = n$ and $s \leq n$, $t_{cals}(n) = n(n-1)/2p$. Each while loop iteration requires 2 communication rounds, of $(p-1)$ communications for each. Thus we have $2(p-1)$ communications by iteration. In the worst case, $k = n$ and $t_{comms}(n) = 2n(p-1)$. Thus PGkAA runs in $\mathcal{O}(n^2/p)$ execution time with $\mathcal{O}(np)$ communication rounds.

Proof of Theorem 3

Proof: The data reorganization preprocessing runs in $\mathcal{O}(m \times n \log(n))$, where n is the number of records, m the number of quasi-identifier attributes, with $m \ll n$. Indeed, for sorting each attribute, the TimSort [42] algorithm is applied.

Each processor performs $(r-1)$ steps to find the minimum merge cost class at each iteration. Thus the execution time for a processor is given by:

$$t_{cals}(n) = (r-1) + (r-2) + \cdots + (r-s) = s(2r-s-1)/2$$

In the worst case, $r = \frac{n}{p}$ and $k = \frac{n}{p}$ (i.e. $s \leq \frac{n}{p}$). We obtain :

$$t_{cals}(n) = \frac{n}{2p} \times \left(\frac{n}{p} - 1\right) + m \times n \log_2(n)$$
$$= \mathcal{O}\left(\frac{n^2}{p^2} + m \times n \log_2(n)\right)$$

As $n \log(n) < \frac{n^2}{p^2}$ (when $n \to +\infty$), thus:

$$t_{cals}(n) \approx \mathcal{O}\left(\frac{n^2}{p^2}\right)$$

An All-to-one communication round is Necessary at the end of the local computation. We get $(p-1)$ communications. Thus, PGPkAA algorithm runs in $\mathcal{O}\left(\frac{n^2}{p^2}\right)$ with $\mathcal{O}(p)$ communication rounds.

Proof of Theorem 4

Proof: The data reorganisation preprocessing runs in $\mathcal{O}(m \times n \log(n))$. The execution time for a block of processors according to theorem 2 and 3 is given by:

$$t_{cals}(n) = \frac{b}{p}\sum_{i=1}^{s}(r-i) = \frac{bs(2r-s-1)}{2p} + m \times n \log_2(n)$$

In the worst case we have: $r = \frac{n}{b}$ and $k = \frac{n}{b}$ (i.e. $s \leq \frac{n}{b}$). Thus, $t_{cals}(n) = \frac{n}{2p} \times \left(\frac{n}{b} - 1\right)$. Since $b \leq \log_2 p + 1$,

$$t_{cals}(n) = \frac{n}{2p} \times \left(\frac{n}{(\log_2 p)} - 1\right) + m \times n \log_2(n)$$

.

So,

$$t_{cals}(n) \approx \mathcal{O}\left(\frac{n^2}{p \log_2 p}\right)$$

Thanks to theorem 2, we obtain $O\left(\frac{np}{(\log_2 p)^2}\right)$ communication rounds in a block of processors. At the end of local computations by each block, there is an All-To-One communication round. Thus, H-PGPkAA runs in $\mathcal{O}\left(\frac{n^2}{p \log_2 p}\right)$ execution time with $\mathcal{O}\left(\frac{np}{(\log_2 p)^2}\right)$ communication rounds.

Appendix C

TABLE V: Overall execution time (in minutes) of parallel algorithms for Adult dataset

Algorithm	k = 2				k = 3				k = 5				k = 10				k = 100			
	Number of processors				Number of processors				Number of processors				Number of processors				Number of processors			
	1	2	8	32	1	2	8	32	1	2	8	32	1	2	8	32	1	2	8	32
PGkAA	44.17	28.9	17.15	14.04	55.9	37.14	22.67	18.88	61.79	41.3	26.48	21.89	64.89	43.7	28.29	23.31	65.01	44.41	27.82	23.59
PGPkAA	44.17	12.02	1.05	0.16	55.9	15.16	1.35	0.19	61.79	16.44	1.45	0.2	64.89	17.2	1.49	0.2	65.01	17.34	1.42	0.21
H-PGPkAA	44.17	11.85	3.89	0.96	55.9	14.84	4.8	1.21	61.79	16.21	5.21	1.28	64.89	17.06	5.75	1.39	65.01	17.09	5.39	1.36

TABLE VI: Overall execution time (in minutes) of the PGPkAA algorithm for Florida dataset

DB size	k = 2				k = 3				k = 5				k = 10				k = 100			
	Number of processors				Number of processors				Number of processors				Number of processors				Number of processors			
	1	32	64	128	1	32	64	128	1	32	64	128	1	32	64	128	1	32	64	128
6×10^4	19.32	0.47	0.24	0.17	27.70	0.50	0.24	0.18	34.37	0.55	0.25	0.18	41.93	0.51	0.26	0.18	46.56	0.51	0.25	0.18
12×10^4	44.38	1.27	0.51	0.27	65.77	1.49	0.54	0.28	82.45	1.49	0.59	0.28	96.41	1.57	0.56	0.29	120.04	1.56	0.57	0.29
24×10^4	150.69	1.32	0.75	0.51	214.90	1.67	0.89	0.55	278.58	1.79	0.96	0.57	381.97	2.01	1.00	0.58	459.55	2.10	1.02	0.61
5×10^5	738.91	5.41	3.67	1.82	1019.90	6.70	4.47	2.07	1541.98	7.84	4.75	2.20	1865.10	8.33	4.93	2.25	2335.05	8.79	4.99	2.25
10^6	3140.39	25.93	13.41	7.17	4742.91	32.61	15.78	7.86	6115.18	37.35	17.25	8.26	8035.01	39.37	17.92	8.45	9608.98	40.19	18.32	8.52

TABLE VII: Overall execution time (in minutes) of the H-PGPkAA algorithm for Florida dataset

DB size	k = 2			k = 3			k = 5			k = 10			k = 100		
	Number of processors			Number of processors			Number of processors			Number of processors			Number of processors		
	32	64	128	32	64	128	32	64	128	32	64	128	32	64	128
6×10^4	0.93	0.69	0.64	1.24	0.90	0.81	1.48	1.03	0.97	1.65	1.13	1.07	1.74	1.18	1.10
12×10^4	1.84	1.33	1.25	2.64	1.92	1.73	3.34	2.34	2.12	3.83	2.65	2.37	4.31	2.96	2.64
24×10^4	6.51	4.60	3.95	9.58	6.77	5.69	12.43	8.66	7.26	14.71	9.97	8.85	17.45	11.18	9.64
5×10^5	40.39	28.61	19.63	60.52	42.03	28.21	79.47	53.71	37.86	93.20	64.56	44.87	102.01	72.30	51.24
10^6	184.45	134.87	86.17	285.73	202.79	135.68	384.04	262.19	170.25	431.29	305.97	201.39	490.02	346.06	233.63

TABLE VIII: Information loss of parallel algorithms for Adult dataset

Algorithm	k = 2			k = 3			k = 5			k = 10			k = 100		
	Number of processors			Number of processors			Number of processors			Number of processors			Number of processors		
	2	8	32	2	8	32	2	8	32	2	8	32	2	8	32
PGkAA	2.28	2.28	2.28	4.14	4.14	4.14	6.5	6.5	6.5	10.11	10.11	10.11	26.98	26.98	26.98
PGPkAA	2.29	2.42	2.56	4.2	4.48	4.8	6.58	7.18	7.82	10.25	11.44	12.55	27.41	31.0	35.43
H-PGPkAA	2.28	2.31	2.37	4.18	4.24	4.36	6.58	6.62	6.91	10.19	10.41	10.88	27.74	28.14	29.43

TABLE IX: Information loss of the PGPkAA algorithm for Florida dataset

DB size	k = 2				k = 3				k = 5				k = 10				k = 100			
	Number of processors				Number of processors				Number of processors				Number of processors				Number of processors			
	1	32	64	128	1	32	64	128	1	32	64	128	1	32	64	128	1	32	64	128
6×10^4	0.24	1.85	2.52	0.35	0.43	3.0	4.06	0.66	0.73	4.47	6.0	1.17	1.31	6.66	9.18	2.21	4.5	24.84	40.09	11.66
12×10^4	0.14	1.38	1.89	0.2	0.26	2.26	3.06	0.38	0.45	3.38	4.53	0.67	0.81	5.1	6.77	1.28	3.27	16.39	25.26	7.46
24×10^4	0.12	0.14	0.16	0.18	0.22	0.26	0.29	0.33	0.38	0.44	0.5	0.58	0.68	0.8	0.93	1.11	2.87	3.76	4.6	5.8
5×10^5	0.12	0.14	0.15	0.17	0.22	0.25	0.27	0.3	0.36	0.41	0.45	0.49	0.65	0.74	0.8	0.9	2.55	3.22	3.7	4.6
10^6	0.12	0.13	0.14	0.16	0.21	0.24	0.27	0.31	0.36	0.41	0.45	0.51	0.63	0.72	0.8	0.91	2.52	2.96	3.35	4.03

TABLE X: Information loss of the H-PGPkAA algorithm for Florida dataset

DB size	k = 2			k = 3			k = 5			k = 10			k = 100		
	Number of processors			Number of processors			Number of processors			Number of processors			Number of processors		
	32	64	128	32	64	128	32	64	128	32	64	128	32	64	128
6×10^4	0.25	0.25	0.25	0.44	0.44	0.45	0.76	0.75	0.77	1.37	1.35	1.37	5.0	5.21	5.12
12×10^4	0.15	0.15	0.15	0.27	0.27	0.27	0.46	0.45	0.46	0.84	0.84	0.85	3.62	3.58	3.65
24×10^4	0.13	0.13	0.13	0.23	0.23	0.23	0.39	0.39	0.39	0.7	0.7	0.71	3.07	3.1	3.11
5×10^5	0.12	0.12	0.12	0.22	0.22	0.22	0.37	0.37	0.37	0.66	0.66	0.65	2.67	2.68	2.65
10^6	0.12	0.12	0.12	0.22	0.21	0.22	0.37	0.37	0.37	0.65	0.65	0.65	2.59	2.59	2.62