Building Privacy-Preservation Models for Distributed Processing Platforms

Sibghat Ullah Bazai

A thesis submitted in partial fulfilment of the requirements for the degree of
Doctor of Philosophy (Ph.D.) in Computer Science,
Massey University, 2020.

Abstract

The widespread proliferation of data collection has increased a serious privacy concern in recent years. Data anonymization approaches have been proposed as a privacy-preserving technique to preserve the privacy of data. However, most existing data anonymization approaches have been designed to work with a small number of dataset within a single machine environment thus often not suitable for big data. To resolve these limitations, many scalable data anonymization solutions that can work with the distributed processing platform (e.g., MapReduce and Spark) has emerged to take advantage of scalability and other supports required for big data. However, due to lack of inherent support for the algorithms involved in data anonymization techniques, these existing proposals often encounter many implementation and performance bottlenecks. In the studies presented in this thesis, we propose a set of novel data anonymization approaches that can work well in the most popular distributed processing platforms for big data such as MapReduce and Spark.

Our first set of studies address the privacy concerns involved in MapReduce platform that processes sensitive data without an appropriate privacy protection which may allow adversaries to break two very important security principals such as data confidentiality and integrity. Firstly, we propose a privacy-preservation platform as an extra layer on MapReduce to provide a set of privacy services to produce different sets of privacy-preserving anonymized datasets that can be safely processed by MapReduce. Secondly, we also offer a privacy-preserving k-NN based classifier for MapReduce. Instead of working with plaintext, our k-NN classifier can work on any anonymized datasets to protect the privacy concern of input data while still providing accurate classification results.

In our second set of studies, we address the concerns in Apache Spark that lack appropriate supports for many popular data anonymization techniques. We first investigate the requirement for the types of support required for many data anonymization approaches which often demand multiple read and write operations. We argue that existing approaches fail to provide supports for caching intermediate data in memory which found to contribute performance degradation. To address this problem, we propose a Resilient Distributed Dataset (RDD) based data anonymization model that avoids expensive disk I/O. We also argue that many existing methods do not provide support for iterative intensive operations which appear in many data anonymization technique such as subtree generalization. We propose a generic approach for implementing subtree-based data anonymization techniques for Spark that provide more effective support for iteration intensive operations. Extending from this, we also provide a novel hybrid approach that can more effectively apply different data anonymization techniques for multi- dimensional data. We argue that our hybrid approach offers much better control for the expensive RDD creation and the size of partitions attached for each RDD which is much better suited to reduce many overheads such as involved in re-computation, shuffle operations, message exchange and cache management.

The experimental studies confirm that our novel privacy-preserving models implemented on both MapReduce and Spark provide high performance and scalability while supporting high levels of data privacy and utility.

I would like to dedicate this thesis to my Parents and Siblings for their endless love, motivation, and encouragement. Especially, this thesis is dedicated for the Memory of My Mother who provided me with invaluable support and encouragement and also instilled three life-changing words; Faith, Consistency and Hard Work.

Acknowledgements

With all praises to Allah Almighty for enabling me to be what I am today.

I would like to thank several people who constantly provided me with advice, help, support, and understanding during the time at Massey University while working on my PhD.

Firstly, I would like to express my deepest gratitude and appreciation to my principal supervisor, Associate Professor Julian Jang-Jaccard. I am honored to have received her supervision. She guided me through the difficulty times that I had I struggled. I gained so much from her in terms of research, writing, and presenting ideas. I will always be very grateful for her guidance, support, and encouragement. I would have not been able to complete this thesis without the kind and patient help from her.

I would like to extend my gratitude to my co-supervisors, Dr. Xuyun Zhang and Professor Ruili Wang. It was a great benefit having their extra comforting and reassuring voices as well as guidance in times when I was uncertain. Their support, patience, and encouragement gave me the confidence I needed to complete my PhD.

I would like to express my gratitude and extend my sincere thanks to Nouman Sarangzai, Sadiq Ahmed, Dr Jan Muhammad, and Syed Aurangzeb shah for being wonderful teachers and mentors.

I would like to extend my sincere thanks to my cybersecurity lab colleague, Dr. Hooman Alavizadeh for his valuable guidance and proofreading my research.

I would like to express my great appreciation to the people whom I have collaborated and built friendships for in the School of Natural and Computational Sciences (SNCS) and cybersecurity lab in the Massey University: Rahila, Faheem, Mehmood, Amir, Junbo, JinJin, Yuanyuan, Azadeh, and Jinting.

Outside of the lab, I am grateful to all those that have supported me throughout this journey, from a warm meal and discussion we shared from the beginning of the day until we go to bed (or couch) towards the end of the day. I am also fortunate to have had the company of many wonderful friends in Auckland. I thank Aymen, Dandachli, Muhammad, Fahad, Fawad, Aziz, Orabi, Saad, and my dearest friend Muhammad Haris for all the quality time we spent together.

I would like to thank the Government of Islamic Republic of Pakistan and the Balochistan University of Information Technology, Engineering, and Management Sciences for their generous financial assistance as well as the experiences they have allowed me to share in.

Last but not least, I thank my immediate and extended family. I would thank my Father for raising me as an independent kid and paying a special attention to my education from the first day of my school life. I pass my special gratitude to my Mother for her endless love and continuous support in my all endeavors; I miss you mom; may you rest in eternal peace. I am forever grateful to my siblings who have helped me throughout my life: suggesting the computing major in my bachelor, sparing time for explaining mathematical concepts to me, planning and arranging my travels, and providing me financial and emotional support when I needed it. Finally, a very special thanks to my Better Half; this thesis would have not been complete without her patience and sacrifice.

Publications Arising from this Thesis

The core of this thesis was based on the following peer-reviewed journals and conferences.

- Bazai, S. U., Jang-Jaccard, J., and Zhang, X. (2017, July). A privacy preserving platform for MapReduce. In International Conference on Applications and Techniques in Information Security (pp. 88−99). Springer.
- Bazai, S. U., Jang-Jaccard, J., and Wang, R. (2017, December). Anonymizing k−NN classification on MapReduce. In International Conference on Mobile Networks and Management (pp. 364−377). Springer.
- Bazai, S. U., Jang-Jaccard, J. , and Zhang, X. (2019). Scalable big data privacy with MapReduce. In Encyclopedia of big data technologies (pp. 1454−1462). Springer. *(Book Chapter)*
- Bazai, S. U. and Jang-Jaccard, J. (2019, December). SparkDA: RDDBased High−Performance Data Anonymization Technique for Spark Platform. In International Conference on Network and System Security (pp. 646−662). Springer.
- Bazai, S. U., and Jang-Jaccard, J. (2020). In-Memory Data Anonymization Using Scalable and High Performance RDD Design. Electronics, 9(10), 1732.
- Bazai, S. U., Jang-Jaccard, J. and Alavizadeh, H. A Generic Approach for Subtree-Based Data Anonymization for Apache Spark. Submitted to IEEE Access. (Second revision) .
- Bazai, S. U., Jang-Jaccard, J. and Alavizadeh, H. A Novel Hybrid Approach for Multi-dimensional Data Anonymization for Apache Spark. Submitted to *Transactions on Privacy and Security (Under Review)*.

Contents

List of Figures

List of Tables

List of Acronyms

AL	Anonymization Level
BUG	Bottom-Up Generalization
CAVE	Average Equivalence Class Size
DGH	Domain Generalization Hierarchies
DM	Discernability Metrics
DP	Differential Privacy
EC	Equivalent Class
ED	Euclidean Distance
GFS	Google File System
GL	Generalization Level
HDFS	Hadoop Distributed File System
IE	Information Entropy
IG	Information Gain
IL	Information Loss
JVM	Java Virtual Machine
KG	K-anonymize Group
KLD	Kullback Leibler Divergence
KNN	K-nearest neighbors
LD	Location Density
LRU	Least Recently Used
MAC	Mandatory Access Control
MAE	Mean Absolute Error

MDSBA	Multi-Dimensional Sensitivity-Based Anonymization
MDTDS	Multi-Dimensional Top-Down Specialization
ML	Machine Learning
NYTWA	New York Taxi Workers Alliance
OLA	Optimal Lattice Anonymization
PG	Privacy Gain
PT	Private Table
QID	Quasi-IDentifiers
RDD	Resilient Distributed Datasets
RMSE	Root Mean Square Error
SA	Sensitive Attribute
TDS	Top-Down Specialization
TT	Taxonomy Trees
UDF	User Defined Function
YARN	Yet Another Resource Negotiator

Chapter 1

Introduction

Processing a huge amount of data using distributed processing platforms (e.g., MapReduce, Spark) has raised a considerable privacy concern. The concern is due to the fact that these platforms distribute the data and operations to third party infrastructure (e.g., cloud). The third-party infrastructure, often not in the administrative control of the data owner, increases the possibility for data breach. Adversaries can gain unauthorized access to the data by monitoring the operations or just by looking at the output stored in the administration domain of clusters. In addition, they can also infer the original input by linking the output with other types of data, for example with non-sensitive data published on social media such as Facebook and Twitter or background auxiliary information gained via a friend or family - referred as composite attack [51].

Privacy concern is raised at multiple levels in both MapReduce and Spark platform. Both platforms process data using multiple mappers and reducers. If one of the many mappers and reducers is compromised by an adversary, it may provide incorrect outputs. An insecure mapper and reducer can be compromised and in turn grant privileges to the adversary to run many malicious activities. Preventing mappers and reducers from being compromised is not an easy task and therefore privacy of an individual cannot be guaranteed [41].

Mappers transform the input original key/value pairs to a set of intermediate key/-value pairs while reducers aggregate to compute and write sets of intermediate values. The output, however, can bring serious privacy concerns. Firstly, the output can directly leak sensitive information because it contains the global view of the final computation. Secondly, the output can also indirectly leak information via composite attacks.

In recent years, more than 4 billion users have suffered from personal information loss as a result of data breach in big data analytic platforms [27, 101, 160].

Data anonymization provides privacy-protection to individual data and has been widely adopted as the practical method to produce privacy-preserving results with high data utility. Privacy can be guaranteed by using privacy-preserving approaches such as K-anonymity [126,127] and its many variations such as t- closeness [86], δ -presence [96], p-

sensitivity [140], Differential Privacy [43] (DP) etc. The choice of which privacy-preserving approach to choose is often application dependent and needs a full understanding of the details of each technique and many (privacy) associated parameters (e.g., K-group size, generalization level) to produce different results [114].

In recent years, researchers have adopted traditional data processing approach in big data processing platforms such as MapReduce [5,7,37,39,61,62,110,121,139,146,149] and Spark [10,25,80,115,119,124,128,158] using many data anonymization based approaches.

However, these platforms were designed without the understanding of the algorithms associated with data anonymization techniques and thus often do not have appropriate supports which result in many implementation and performance overheads.

1.1 Research Goals

We first investigate the most widely used existing techniques of privacy preserving mechanisms that utilise K-anonymization and others (e.g., differential privacy). Our goal is to examine these techniques in depth to understand which aspects of these techniques do not work well when implemented in either MapReduce or Spark. We also examine different architectural aspects of MapReduce and Spark to understand what types of algorithms are supported better or not. By understanding the combination of the most important parts of privacy preserving mechanisms (i.e., data anonymization techniques) and the strengths and weaknesses of the distributed processing platform, we propose a new set of privacy-preserving models that provide high performance with scalability while supporting for a high level of data utility and privacy.

1.2 Research Questions

With the research goal we described earlier, we attempt to provide answers to the following research questions in this thesis.

- Q1: How data anonymization approaches can be used to preserve the privacy of data while processing data in a distributed platform?
- Q2: How to prevent adversaries from accessing sensitive information during the execution of data mining algorithms such as classification and aggregation in a distributed processing platform?
- Q3: What are the architectural limitations and performance bottlenecks of the existing distributed processing platforms that provide various data anonymization solutions?
- Q4: What factors (aspects) need to be considered for iterative intensive data anonymization techniques to run well on a distributed platform while still maintaining high data utility?

- Q5: How to avoid the limitations of the recursive and iterative based data anonymization approaches associated with multi-dimensional features?

1.3 Research Contribution

We aim to develop a set of privacy preservation models that can work well in the popular distributed processing platform such as Spark and MapReduce. Our main contributions are as followed:

- We build a privacy preserving platform for MapReduce which offers a set of services to produce different types of privacy-preserving anonymized datasets depending on different application needs. We provide the feasibility of our platform by producing two different anonymized datasets that utilize the K-anonymity and differential privacy based on the real-life dataset using NYC Taxi dataset.

- We offer a privacy-preserving k-NN classifier which provides a classification service on anonymized datasets without revealing anything about input data. We evaluate the relationship between the privacy parameters (e.g., K group size, sensitivity and privacy budget) and the quality of classification results using two case studies of the k-anonymity and differential privacy.

- We propose a scalable and high performance RDD-based data anonymization strategy for Spark, named as SparkDA. Our proposal is based on Spark's Resilient Distributed Dataset (RDD) with two critical operations of RDDS, FlatMapRDD and ReduceKeyByRDD, respectively. We demonstrate that the RDD-based approach utilises in-memory operations more efficiently which avoids expensive disk I/O. This approach is better suited for many data anonymization algorithms where iterations occur heavily during anonymization processing.

- We provide a generic approach for subtree-based data anonymization for Spark. Any subtree-based algorithms (e.g., top-down, bottom-up, hybrid) can take the advantage of our approach and can be implemented to work better for Spark platform compare to MapReduce-based counterparts. Our approach offers more effective partition management, improved memory usage that utilises cache for frequently referenced intermediate values, and enhanced iteration support.

- We propose a novel hybrid approach for multi-dimensional data anonymization for Spark. Our proposal can produce more fine-grained privacy-preserving data anonymization results on Spark which allows to apply a different anonymization strategy applied for each dimension of data based on Mondrian algorithm. Our approach avoids the creation of too many RDDs that were often observed in the existing recursion-based approach while it also produce much smaller partitions

compare to the existing iteration-based approach. We illustrate that our hybrid approach is effective in avoiding many overheads involved in re-computation, shuffle operations, message exchange and cache management.

1.4 Thesis Organization

The organization of this thesis is as follows.

In Chapter 2, we address composite attack that exploits the output data obtained without authorization in the clusters-based MapReduce. We propose a generic privacy-preservation strategy to prevent privacy breach for the data that is processed by MapReduce. Our approach performs anonymization in reducer phase to produce sanitize outputs while maintaining a high data utility. We demonstrate that the application of our platform on two state-of-the-art privacy preserving mechanisms, K-anonymity and Differential Privacy (DP), respectively, to illustrate the feasibility of our proposed strategy to use in real-world applications. (The work has been published in [24]).

In Chapter 3, we identify the privacy concern of MapReduce that processes data in plain text which pose a serious privacy risk. We propose a novel privacy-preserving k-NN classifier algorithm that can run a classification task on different anonymized datasets without exposing sensitive information. Two separate sets of anonymized data are produced by utilising K-anonymity and Differential Privacy (DP) approaches. We illustrate that different classification results are obtained based on the different sets of anonymized datasets. We provide the impact in the trade-off between the level of privacy protection and the high-value insights on the classification between basedline and anonymized datasets (The work has been published in [23]).

In Chapter 4, we propose a new novel data anonymization technique for Apache Spark named "SparkDA". (The earlier version of this work has been published [19]). The SparkDA takes the full advantages of innovative Spark features, such as better partition control, in-memory process, and cache management for iterative operations while providing high data utility with privacy. Our data anonymization algorithms are implemented in two main data processing RDD transformations, FlatMapRDD and ReduceByKeyRDD, respectively. Our experimental results show that our proposal provide high-performance, scalability and better memory/cache optimization while providing required data privacy and utility. The extended version of this work is published in MDPI journal of Electronics [20].

In Chapter 5, we identify shortcomings of supporting iteration intensive algorithms such as subtree generalization in Spark. We propose a generic approach for implementing subtree-based data anonymization techniques for Spark. Our RDD-based approach resolves the limitations of MapReduce-based approaches by offering effective partition management, improved memory usage that utilizes cache for frequently referenced intermediate values, and enhanced iteration support. Our experimental results show that our

proposal provides better performance compare to other similar methods while maintaining high levels of data privacy and utility. (The work has been submitted to IEEE Access journal [21]).

In Chapter 6, we propose a novel hybrid approach for data anonymization techniques that deal with multi-dimensional features based on Mondrian. Our hybrid approach creates a lot fewer numbers of RDDs compare to the existing recursion-based approach and the smaller size partitions attached to each RDD compare to the existing iteration-based approach. These contribute towards reducing many overheads involved in re-computation, shuffle operations, message exchange and cache management. The experiment results illustrate that our proposal outperforms other similar methods and can be used as a scalable, high-performance, and privacy-preserving data anonymization solution for big data. (The work has been submitted to the journal of Transactions on Privacy and Security (TOPS) [22]).

In Chapter 7, we conclude the thesis and illustrate some future research directions.

Chapter 2

A Privacy Preserving Platform for MapReduce

Summary

Big data applications typically require a large number of clusters, running in parallel, to process data fast and more efficiently. This is typically controlled and managed by MapReduce. In MapReduce operations, Mapper maps input original key/value pairs to a set of intermediate key/value pairs while Reducer reduces a set of intermediate values, computes and writes the output. This output however can bring serious privacy concerns. Firstly, the output can directly leak sensitive information because it contains the global view of the final computation. Secondly, the output can also indirectly leak information via composite attacks where the adversary can link the final output with public information published via different sources such as Facebook or Twitter. To address such privacy concerns, we propose a privacy preserving framework which can prevent privacy leakage in MapReduce. Our framework can be plugged into the Reducer phase to sanitize the final output in such a way that the privacy is preserved while it yet provides a high data utility. We demonstrate the feasibility of our framework by providing empirical studies and highlighting that our proposal can be used for real world applications. The study illustrates the concrete examples of applying two state-of-the-art privacy preserving mechanisms, K-anonymity and differential privacy (DP), respectively, on the New York Taxi dataset.

2.1 Introduction

Tremendous work in data analytics has made an impressive progress to assist in critical data-driven decision-making processes [3]. What makes scalable data analytics possible is the emergence of MapReduce which provides a parallel computing paradigm for big data applications [40]. The name MapReduce comes from its two main functions: map and reduce. In the map function, input data is usually spilt into smaller chunks and computed in a completely parallel manner by independent cluster nodes. The reduce function consolidates the smaller chunks into a group. The several groups from different cluster nodes are computed and are written as an output. The MapReduce platform provides libraries which can provide everything meant for computing on larger clusters from parallelization, data distribution, load balancing, and fault tolerance [4, 118].

One considerable privacy concern raised in the way the reducer handles the out-put and writes it to the file system. The concern is raised because the reducer often runs on third-party infrastructure (i.e., public clouds). The administrators of the third party (or adversaries) can easily infer the sensitive knowledge simply by directly examining the output files. Or indirectly, they can also infer the original input by linking the output with other types of data, for example with non-sensitive data published in social media such as Facebook, Twitter or background auxiliary information gained via a friend or family - this is known as composite attacks [132]. To solve such privacy concern, data anonymization techniques have been used such as through simple data masking, grouping, K-anonymity [127], t-closeness [86], l-diversity [89], differential privacy (DP) [43]. However, these techniques exist in isolation from each other in which often are tailored to address a specific problem for a specific domain.

We proposed a privacy-preserving platform to prevent MapReduce privacy leakage. Our platform is designed in such a way that it can accommodate many different privacy-preserving mechanisms and corresponding algorithms that can implement different strategies for different data anonymization results. Our platform can be plugged in the reducer phase to sanitize the final output so that it can prevent adversaries to inference the original data or other privacy related data within dataset. We demonstrate the feasibility of our platform by providing empirical studies which aim to highlight that our proposal can be used for real life application. The studies illustrate the concrete examples of applying two state-of-art privacy-preserving mechanisms, K-anonymity and differential privacy respectively along with New York Taxi dataset in our platform.

The rest of chapter is organized as follows. In section 2.2, we describe the major technologies and their important features involved in this study. In section 2.3, we describe the related work that provides different types of solutions to addressing MapReduce privacy problems. In section 2.4, we formulate our proposed framework in detail. In section 2.5, the results of our experiment analysis based on our empirical study are demonstrated. Finally, we conclude in 2.6 our work and discuss some future directions.

2.2 Background

2.2.1 MapReduce

The MapReduce have been a critical technology in processing big data analytics. MapReduce was originally proposed by Google [40]. As a typical data batch processing technology, its applications have been developed for the fields in data mining, machine learning, data analytics and other fields. Due to its powerful parallel processing support, MapReduce has become the key technology for data processing [54].

Big Data processing is typically performed by feeding a large dataset to mappers that split the data into smaller more manageable chunks for different nodes of clusters. Mapper is responsible for reading each data, line by line, and saving that each assigned information into key/value pairs where the key is the data from the input file and the value is the number of times that the key appears in the data. After completing this process, mapper stores the key/value pairs in a temporary location. The temporarily located data is then processed using shuffle and sort then forwards this intermediate value to a reducer. The reducer performs the collective combining job, that is, to collect all intermediate data with the same key/value pairs and store them into HDFS.

2.2.2 K-anonymity

K-anonymity is the first data anonymization technique with formal mathematical support as a proof. Sweeney [126] introduced K-anonymity in 2002 by stating that without ensuring K individuals in aggregation a single aggregate statistic should not be published. In his definition, Quasi- Identifiers (QID) are attributes in dataset which may be linked from publicly available dataset. The main goal to achieve K-anonymity is to replace QID values with more general values, for example generalizing 3 different values “15”, “17”, and “19” into a more general single value “15-20” [125].

K-anonymity is considered as one of the most popular techniques thus has been studied well in the data anonymization community. In typical processing of K-anonymity, it utilizes two distinct techniques known as generalization and suppression with the aim to decrease the granularity of quasi identifier. Using generalization, more granular values are combined together to create a broader category. This can be achieved both for numerical variables (e.g., number of passenger in single taxi 3, 4, and 5 into a broader category of 3-5) and for categorical variables (e.g., generalizing pickup time data from ”2013-08-07 17:38:43“to ”2013-08-07”). Generalization replaces the original record attributes with less exact but constant values. QIDs may become generalized to a certain point where a few conclusions can be drawn about their relationships with other records. However, caution should be taken as repeated generalization could decrease the quality of the entire dataset. Suppression works differently from generalization by removing any records that violate anonymity standards from the dataset entirely. Also, caution should be taken that

suppression can skew the integrity of dataset when values are eliminated disproportionately to the original distribution of the data. More often than not, suppression is used in conjunction with the generalization to improve the anonymization efficiency, for example, the records that were not within the boundary of K-anonymity after generalization can be automatically suppressed [127].

2.2.3 Differential Privacy

Formally differential privacy DP [43] can be defined as; if datasets D_1 and D_2 are only differ from a single record the function f over the range of output R is ϵ - Deferentially private [44] for all subset S_b of R by satisfying the following condition

$$Pr[f(D_1 \in S_b] \leq e^{\epsilon}.Pr[f(D_2) \in S_b], \tag{2.1}$$

DP ensures that output will not raise the probability of any adversary learning any individual data by more than the factor. To measure the perturbation in any mechanism's sensitivity plays a vital role. Mainly two types of sensitivities are measured; global sensitivity and local sensitivity. The global sensitivity G_s [70] is considered as an essential notation of DP noise calculation and defined as maximal differences between query results on neighbouring datasets and indicates how much the difference should be hidden in mechanisms. Local sensitivity calibrates the record-based differences between query results on neighbouring datasets and also satisfies the DP [45].

Privacy budget- ϵ controls privacy guarantee level of any Mechanism. For types of applications which require higher degree of privacy, the lower privacy budget is more efficient which can range from 0.001 to max 1. The ultimate privacy guarantee depends on the step with the maximal ϵ . Laplace and Exponential mechanism are the two most common mechanisms to provide noise for DP.

Laplace mechanism adds controlled noise to the query result before returning it to the adversary [148]. The noise is generated using Laplace distribution, typically applied in the continuous data and it controls the random noise by Laplace distribution [159]. Let a function f: $D_1 \implies R$ over a dataset D_1, the Laplace mechanism is used to achieve ϵ -differential privacy.

$$M(D_2) = f(D_1) + Lap\frac{\Delta f}{\epsilon}, \tag{2.2}$$

Where Δf represents the sensitivity of query f . For non-numeric queries exponential mechanism perform much better then Laplace mechanism [30], the formal definition of exponential mechanism is: Let E:(D,Ψ) be a quality function of dataset D that measures the score of output $r \in R$ Then an Exponential mechanism is M_c is ϵ-differential privacy if

$$M_c(D) = Return\ r\ with\ the\ Probability\ \alpha \exp \epsilon \frac{E(D_1, r)}{2\Delta f}, \tag{2.3}$$

Where Δf represents the sensitivity of query f.

2.3 Related Work

In this section, we describe existing works that discuss security and privacy issues in MapReduce in different stage of its operations.

A number of works in this area focuses protecting the intermediate values that are produced after mapper function [37,78,148]. The intermediate values which are stored in a temporary file by MapReduce platform are not supported with any protection mechanism [32] therefore these values can be easily accessed by adversaries. The deletion of the intermediate (i.e., temporary) files happens at the end of mapper and reduce job [100]. Pig allow user to run high level scripting language on MapReduce platform in the Hadoop ecosystem. Pig does not handle temporary files deletion if the script execution failed or killed. Once this happens, the deletion task is left to the developers to handle it on their own without any support from the MapReduce platform. The authors in [32] discuss three main challenges of MapReduce when used in cloud platform: Scalability and Dynamic, Cost effectiveness and Data utility and Compatibility. Zhang et al. [155, 157] addresses these issues by proposing a privacy-preserving layer over MapReduce, which satisfies privacy demands itemized by data publishers built on diverse MapReduce privacy models.

Table 2.1: MapReduce and Spark based anonymisation technique on machine learning and data mining algorithms

Algorithm	**Domain**	**Anonymisation Technique**	**Data Processing Platform**
Decision Trees	Classification	*DP* [37] , K-anonymity [126]	MapReduce
Support Vector Machines	Classification	*DP* [146]	MapReduce
Naive Bayes	Classification	*DP* [110]	MapReduce
Support Vector Machines	Classification	*DP* [61]	MapReduce
Expectation Maximization	Clustering	K-anonymity [7]	MapReduce
Micro aggregation	Clustering	DP, K-anonymity [121]	MapReduce
Fuzzy C Means	Clustering	K-anonymity [139]	MapReduce
Random Forest	Clustering	K-anonymity [5]	MapReduce
K-Nearest Neighbors	Classification	K-anonymity [146]	MapReduce
K-Means	Clustering	*DP* [62]	MapReduce
Locally Weighted Linear	Regression	*DP* [39]	MapReduce
Gaussian Discriminant Analysis	Regression	*DP* [70]	MapReduce
Logistic Regression	Regression	*DP* [112]	MapReduce
Aggregation	-	K-anonymity [10]	Spark
Aggregation	-	K-anonymity [119]	Spark
-	Classification	K-anonymity [94]	Spark
Logistic Regression	Regression	K-anonymity [79]	Spark
K-Means	Clustering	K-anonymity [28]	Spark
K-Means	Clustering	K-anonymity [120]	Spark
Ball-tree	clustering	K-anonymity [34]	Spark
Hierarchical Analysis Clustering Tree	Clustering	K-anonymity [73]	Spark
-	Classification	K-anonymity [108]	Spark
Aggregation	-	*DP* [63]	Spark
K-Means	Clustering	*DP* [52]	Spark
Aggregation	-	K-anonymity [107]	Spark
K-Nearest Neighbors	Classification	*DP* [137]	Spark

The existing schemes have certain limitations such as inefficiency to handle incremental data, time taken to update records is more, experiences poor scalability, absence of parallelization, poor execution time, higher rate for loss of information and inequity between data utility and anonymization. MapReduce and Spark framework are supported by a variety of algorithms using two widely used anonymisation techniques such as DP and K-anonymity to solve problems in different domain as illustrated in Table 2.1.

More closed work to us which provides the protection on the reducer is described by Airavat [110] which proposed a secure framework for MapReduce by defining mandatory access control (MAC) with DP on secure Operating system SELinux. Airavat MAC is activated when privacy leakage exceeds from define limit, ensuring high utility and privacy. However, Airavat add pre-configured noise for query which limits its application. Tran and Sato [133] addresses Airavat limitation by allowing users to write reducer code by modifying System's access control, however, if adversary manage to sneak reducer code by changing user right as a trusted user, the proposed solution fail to provide privacy guarantee.

2.4 Proposed solution

In this section, we describe the details of our proposed platform including the major components and their responsibilities.

2.4.1 The proposed framework

We propose a privacy-preserving platform that works collaboratively with mapper and reducer in such a way that it hides the details of the final output by providing mechanisms for various data anonymization while it still provide a high data utility. The approach taken by our platform essentially provides a better flexibility of executing different privacy-preserving mechanisms and corresponding algorithms while ensuring any part of data is not leak during any MapReduce operations. The proposed platform is illustrated in Fig. 2.1.

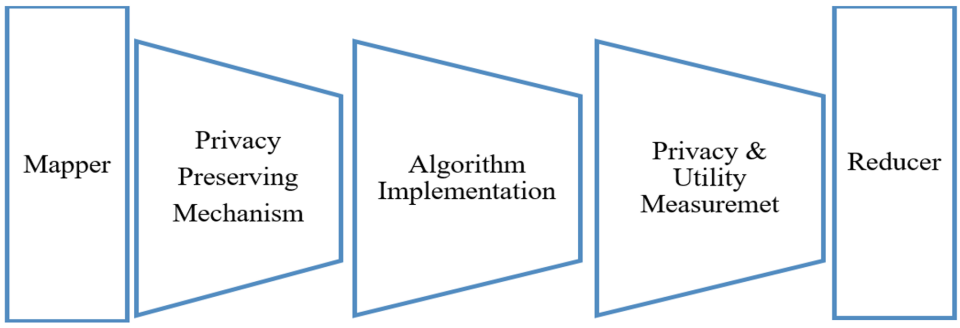


Figure 2.1: Proposed Framework

Our framework consists of three components: Privacy Preserving Mechanism layer, Algorithm Implementation layer, and Privacy-utility measurement layer. The privacy-preserving mechanism layer can accommodate many state-of-art data anonymization mechanisms. Once this mechanism is determined, an algorithm which can produce a specific sanitized dataset can be implemented by the algorithm layer. Finally, Privacy and Utility Measurement layer can measure the different privacy and utility values using different formulas in order to verify the sanitized data is still ensuring the privacy and yet provide enough useful information. The only consideration we made is that the adversary cannot change the behaviour of mappers, nor he can access complete intermediate data.

2.4.2 Components

In this section, we provide the details of the components in our proposed model.

Privacy Preserving Mechanism layer:

This layer receives data from trusted mapper(s) and defines a privacy protection mechanism to preserve the privacy from unprotected reducer and dishonest system administrator. The list of privacy protection mechanisms which can be applied in this layer include for example but not limited to, K-anonymity, l -diversity and DP. Each of these mechanisms has their own strengths and weaknesses and are most often applicable to use in a specific use case. Providing a single privacy-preserving technique is often too limiting to accommodate many different applications scenarios. In our proposal, we address this limitation by allowing the data scientist to decide and pick a mechanism that best suitable for different data anonymization.

Algorithm Implementation layer:

Algorithms deal with the process of transforming the original data into a sanitized data under the umbrella of the privacy-preserving mechanism that was defined in the privacy-preserving mechanism layer. This layer allows the data scientist to choose suitable algorithm for data transformation, algorithm includes, but not limited to; Naïve Bayes, Gaussian Discriminant Analysis, Random Forest, K-means, Maximization, Decision Trees, and many more. Each of these algorithms are used by researcher to achieve anonymization using MapReduce Platform.

Privacy and Utility Measurement layer:

This layer is responsible for providing a functionality where data scientist can measure the privacy and utility trade-off after the original data has been transformed into a sanitized one.

Data utility can ensure that the data still contain enough information where data analytics can still find the relationships and correlations between data. This can be done by a utility measurement (e.g., $RMSE$, MAE). The utility measure can be done in many different ways depending on the type of algorithms it dealt with. For example, to measure data utility in the aggregation, the data utility can be measured by comparing the accuracy of answering aggregate queries between the original dataset and the sanitized

data. For a classification algorithm, the utility measure can be done by comparing the percentage of samples that are correctly classified between the original dataset and the sanitized version. A noise added statistical utility measure can be achieved by calculating the total variance of perturbed data or calculating the length confidence interval of the estimator. Additionally, utility measure can be achieved by using the techniques such as generalization height, privacy information loss ratio, workload aware anonymization. Each approach has its own unique way for calculating a data utility.

Privacy measure can ensure that the data is protected from any privacy attack after the data has been sanitized. The privacy measure is typically done by measuring the uniqueness of data (e.g., the number of unique data). For example, record-based privacy measure, it can measure the ratio of counts that are related to unique record before and after transformation.

2.5 Empirical studies

This section provides an empirical study to illustrate the way to utilise our framework for a real-life application. To this end, we choose two state-of-the-art privacy-preserving mechanisms, K-anonymity and differential privacy (DP), respectively, to demonstrate the effectiveness and usefulness of our proposed framework along with the widely used New York taxi data [42]. The experiments are performed on Intel(R) Xeon(R) CPU E5-1650 v3 @ 3.50GHz, 3501 MHz, 6 Core(s), and 12 Logical Processor(s) with 4 Tarabytes hard drive and 32 GB of RAM. As to the New York Taxi trip data, we specifically use the 2013-2016 version of this dataset. The original New York Taxi trip data includes 19 features in total. We specifically use a subset of the original dataset and only consider four commonly used features: pickup date time, pickup longitude, pickup latitude, and total amount, in order to focus on the discussion of our main idea in this chapter.

2.5.1 Applying K-anonymity on Aggregation Algorithm

For our first empirical study, we used K-anonymity as a privacy preservation mechanism to anonymize the total taxi fare. For an algorithm implementation, we use the following pseudo-code which calculates the approximate total taxi fare within the group size K.

The aggregation algorithm is applied in the following way. We first calculate the average value against the total taxi fare (i.e. Total amount) within all records in the same group. The average becomes a new sanitized value for all records.

To see the effects of the accuracy against different levels of group size K threshold, we use K to be $K \in 5$, 10, 100 and 1000. To understand the effect of anonymized value in the different group size, our experiments carried out again on different location density scale: LD-10, LD-100, LD-1000, LD-2500, LD-5000.We use Root Mean Square Error ($RMSE$) to understand the overall privacy verses utility trade-off based on different K group

Algorithm 1: K-anonymity on Aggregation Algorithm

INPUT: K group size as KG, Location Density as LD, Total Amounts as TA,
OUTPUT: Root Mean Square Error as $RMSE$
ALGORITHM:
Initialize all variables KG, LD, AVG
1: **for each** $i \in K$ **do**
2: Calculate the averages of all $K[i]$.
3: **for each** average **do**
4: $Noise[i] = Avg - KG[i]$;
end for each
5: Repeat Step 2 for every LD
6: **for each** element in n of $Noise$ **do**
7: **for each** element j of TA **do**
8: $RMSE\ [n] = RMSE\ [j] + (Noise[i] - [j])$;
end for each
9: RMSE[n]=RMSE[n]/LD;
end for each
end for each
10: Output $RMSE$;

size and the data size (i.e., location density). The $RMSE$ calculation on different group threshold K on different LD is shown in Fig.2.2.

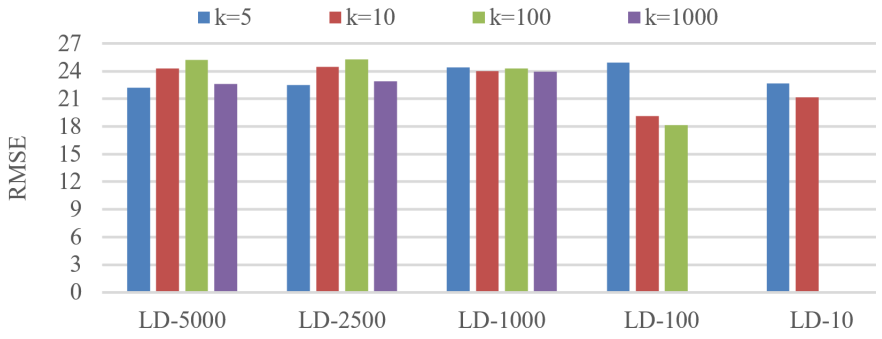


Figure 2.2: Privacy utility trade-off using $RMSE$ on K-anonymity

The results in Fig.2.2 show differences on the group size K for different location densities. We make the following observations;

- The K group size 5 and 10 do not make much difference in the location density LD-10 which illustrates that the distribution of data within 10 records looks to be pretty uniform.
- For the location density LD-100, there is a huge difference in $RMSE$ values among different K group sizes. It indicates that the distribution of data in this group is much more skewed. For example, total amount may vary from minimum 0.16 to maximum 375 which have huge impact on average, due to this the $RMSE$ for each K group vary from 24.91 to 18.15.

- For the location density LD-1000, we experience a good consistency with $RMSE$ values across different K group sizes. It is evident that the distribution of data in this group is unvarying. From the total amount we observe LD-1000 contain even maximum and minimum value for their respective group, due to this we observe even distribution of $RMSE$ for all K group.
- From the location density above 2500, the graphs started looking similar in their $RMSE$ value proportion across different K group sizes. The smaller group size K (i.e., K=5) gives the lowest $RMSE$ value due to its frequent average sum applied across the dataset. Similarly, the largest group size K (i.e., K=1000) also gives relatively low $RMSE$ value because there is more chance for the data to be normalised.

2.5.2 Applying Differential Privacy (DP) on Aggregation Algorithm

Let's suppose a data scientist wants to execute a query to find the average of total amount of taxi fare charged to passengers from JFK airport New York see how much the driver earns from this location collectively in the year between 2013 and 2016. Typically, New York Taxi Workers Alliance (NYTWA) Repair Services have the details of every taxi visited for JFK airport. The NYTWA could use this knowledge to link with non-sensitive published results to deduce the exact salary of a particular driver. To prevent such privacy violation from occurring, rather than using a real value (i.e., the total exact amount of taxi income), it would protect privacy better if a statistically approximate of the total taxi fare is used instead. With that goal in mind, we implemented DP as a privacy preservation mechanism to see the effect of using a noisy mechanism-based technique to provide privacy. For an algorithm implementation, we utilise an aggregation scenario.

Algorithm 2: Differential privacy on Aggregation Algorithm

INPUT: Privacy Budget as PB, Location Density as LD, Sensitivity Sen, Total Amounts as TA
OUTPUT: Root Mean Square Error as $RMSE$
ALGORITHM:

```
1:  for each i ∈ PB do /* outer loop */
2:      initialize Actual Sum = 0
3:      Sanitized Sum SS = 0;
4:      Calculate Noise using Lap(Sen/PB[i])
5:      for each j ∈ TA do /* inner loop */
6:          Add TA[j] to AS
7:          Add TA[j] and Noise to SS
        end inner for each
    end outer for each
8:  Calculate RMSE[i] to be sqrt (square((SS-AS)/LD))
9:  Output RMSE;
```

For the implementation of adding the controlled noise for our algorithm, we use the Laplacian mechanism which generates the random noise in terms of the Laplace distribution from equation (2). This Laplace noise is added to each raw data (i.e., individual total

amount). The value with noise then is summed. To understand the effect of noise in different data size and distribution, our experiments carried out on different Location Density (LD) is used for five different scales: LD-10, LD-100, LD-1000, LD-2500, LD-5000, respectively where the number indicates the size of the data for example, LD-10 represents the 10 pickup locations in the sample. Our experiment study used a fixed sensitive = 1. For each location density, we apply four different privacy budgets (i.e., ϵ(0.001,0.01,0.1 and1) to understand the effect of noise between the privacy budget and the data size. The overall privacy verses utility trade-off, based on different privacy budget and the data size, is calculated using the Root Mean Square Error ($RMSE$) which measures the difference between the raw data and the sanitized data (i.e., the noised injected raw data). Assuming n test samples with raw data values $y_1, \ldots, y_n$ and sanitized values $y_1^*, \ldots, y_n^*$, the $RMSE$ is then given by:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i^* - y_i)^2} \tag{2.4}$$

The $RMSE$ calculation on the privacy budget on different LD is shown in Fig.2.3.

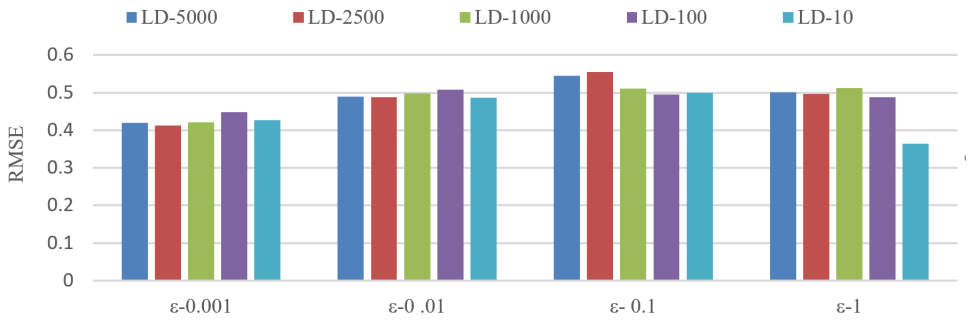


Figure 2.3: Privacy utility trade-off using $RMSE$ on Differential privacy

The results in Fig.2.3 show differences on the privacy budget value ϵ for different location densities. We make the following observations;

- For the privacy budget ϵ-0.001, it provides the lowest error rate denoted by the smallest $RMSE$ values. Most likely, the noise is relatively small compared to other privacy budget. This privacy budget may be applicable for the types of applications where it requires relatively high accuracy, but privacy is not a main concern.
- The privacy budgets ϵ (0.01 and 1) illustrates similar $RMSE$ values no matter the data size. We see a big difference in $RMSE$ result on the LD size 10. This is most likely that the LD size 10 is too skewed to get any meaningful value.
- Using the privacy budget ϵ -0.01, it provides the most uniform distribution of the error rate regardless the different size of location density. It also demonstrates the

highest $RMSE$ values which mean that the most noise was introduced. This privacy budget may be applicable for the types of applications where it requires relatively high privacy, but accuracy is not a main concern.

2.6 Conclusion

In this chapter, we have proposed a privacy-preserving platform for the prevention of privacy leakage in MapReduce by adding three middle layers between mappers and reducers. The novelty of our Platform is that it allows the users to choose anonymization technique in Layer 1, algorithm to process that technique in Layer 2 and utility-privacy trade-off measurement to verify the impact of algorithm and anonymity technique combination in Layer 3. We have presented an empirical study on NYC taxi data using our Platform to illustrate the feasibility and practicability of our proposal. The first empirical study was carried out on K-anonymization with generalization and suppression algorithms with the Precision score to measure the privacy and utility trade-off. The second empirical study was carried out using differential privacy with an aggregation algorithm while we used $RMSE$ as a privacy and utility metric. In future, we plan to provide more privacy preservation mechanisms other than the two we demonstrated. We also plan to develop more real application case studies where it uses not only aggregation, but it also uses classification and feature extraction and consequently different measures of privacy and accuracy trade-offs. Currently we manually set important values to decide the privacy verses utility measures such as K-threshold in case K-anonymity or the privacy budget epsilon and sensitivity value in the case of differential privacy. We plan to enable our platform to automatically pick up these values in the future, depending on the application scenarios.

Chapter 3

Anonymizing k-NN Classification on MapReduce

Summary

Privacy is one of the emerging concerns in big data analytics and its applications. MapReduce platform has been hailed as a solution to provide a number of services which can process big data fast and efficient using its cluster-based nodes in parallel. However, MapReduce can be vulnerable from composite attacks when its output is compromised and combined with public data. To address the limitation, MapReduce must employ a privacy protection mechanism to protect the data being processed by MapReduce operations. Data anonymization has been seen as a viable privacy protection mechanism as it can hide the original data while it still provides necessary queries required for analytics. However, there has not been much study as how to implement data anonymization techniques for MapReduce. In this Chapter, we describe the details of implementing two popular data anonymization techniques on k-NN classification algorithm for MapReduce operations, namely K-anonymity and differential privacy. We describe Map and Reduce algorithms to produce different sets of anonymized data for k-NN classifier. We also illustrate the details of experiments we performed on the multiple anonymized sets. The experiments on k-NN classifier gives an opportunity to examine the effects of privacy vs utility trade-offs of anonymization techniques and its parameters which are often used to fine-tune the level of anonymization.

3.1 Introduction

In recent years, we witness big data containing a huge amount of personal data such as seen in the data acquired by government administrations, health insurances, social networking sites and IoT devices, to name a few. This exponential growth of big data has demanded a requirement for a system which can provide powerful computation and other

related technologies. A big data processing framework using distributed environment, such as MapReduce and Spark, has been widely used to handle such computation to find insights such as correlation between large datasets using Machine Learning algorithms [92, 102, 105].

In Machine Learning, classification algorithms play an important role in data mining to identify a category of a new observation of data into a set of predefined classes or groups. The k-nearest Neighbour (k-NN) [38] is one of such classification methods. In recent years, we witness the adoption of k-NN algorithm in distributed environments to overcome the computational intensity of having to compare distances of every single training data point.

However, processing k-NN in MapReduce raises a number of security issues [137, 146]. In MapReduce, Mappers transform the original input key/value pairs into intermediate key/value pairs after some calculation while reducers aggregate the intermediate values, compute and write them to an output file. These operations at different stage of MapReduce operations are done on plain text which is vulnerable from unauthorized access that puts users' data at a privacy risk. The unauthorized privacy attack can either directly leak sensitive information or indirectly leak information via composite attacks where the adversary can link users' data, illegally obtained at various stage of MapReduce, with public information available via different sources such as Facebook or Twitter.

Providing privacy guarantee during computations of sensitive data can be achieved using privacy preserving techniques such as K-anonymity [127]. K-anonymity uses generalization to hide individual features (also called attribute) or records (also called as tuples) within a crowd or suppression to remove highly sensitive records. The size of crowd is typically determined by a privacy parameter K group size. The use of K-anonymity Differential Privacy DP in MapReduce platform to provide a certain level of privacy are found in [24, 110, 150, 156]. However, these existing studies do not illustrate how to process a privacy preserving technique such as K-anonymity and Differential Privacy DP to be applied in different data analytics scenarios such as classification.

Extending from our earlier [24] study where we illustrated how to apply a K-anonymity and Differential Privacy (DP) in an aggregation scenario [24], this time we illustrate how one can apply a K-anonymity and DP in a classification scenario that utilizes k-NN algorithm. We propose a k-NN classifier which can run on an anonymized data in the MapReduce platform. To the best of our knowledge, our proposed algorithm in this chapter is the first attempt to address the classification implementation on anonymized data in the MapReduce platform. Our main contributions are;

- We illustrate the details of the k-NN classifier algorithms that can run on an anonymized dataset.
- We demonstrate that it is possible to generate different sets of anonymized data using varying degree of privacy parameters (i.e., K group size, sensitivity and privacy

budget) either applied in the different number of features or the different number of records in the K-anonymity and DP algorithms.

- We illustrate that different classification results can be obtained based on the different sets of anonymized datasets.
- We provide the impact in the trade-off between the level of privacy protection (data privacy) and the high-value insights (data utility) on classification before and after different anonymized data.

The rest of the chapter is organized as follows. In Section 3.2, we provide the necessary background knowledge. In Section 3.3, we describe the related work. In Section 3.4, we describe the details of data anonymization strategies we use and explain the algorithms needed for Map and Reducer operations. In section 3.5, the experiments and results are discussed. Finally, we conclude our work and discuss the future work planned ahead of us in Section 3.6

3.2 Background

The k-nearest neighbour method (k-NN) is one of the most widely used classification algorithm in machine learning. Cover and Hart in 1967 formally introduced the original idea of k-NN and its properties [38] . k-NN works directly on the actual instances of the training data as it does not require building a model to represent the underlying statistics and distributions of the original training data [38]. k-NN is based on learning by analogy, that is, by comparing a given test record with training record sets.

Euclidean Distance (ED) is often used to measure the distance of two records where the distance indicates the degree of difference (i.e., if ED is small the two records are likely to be similar while two records are different if ED is big). The distance measure based on equation 3.1:

$$D(X,Y) = |(|X - Y|)| = \sum_{i=1}^{p}(X(i) - Y(i))^2, \tag{3.1}$$

where $X(i), Y(i)$ are the ith dimension attribute values of vector X, Y respectively. In k-NN classification, an output can be seen as a class membership as an object is classified by a majority vote of its neighbours. Thus, a class is typically assigned to the object based on the most common classes observed among its neighbours.

There are many different ways to implement k-NN algorithms including where the classification should be performed (e.g., [122] proposed the centralized paradigm where the k-NN join is performed on a single centralized server) and looking into improving performance overheads (e.g. Parallelization of k-NN algorithm [122]). Especially, many existing approaches have been criticized as the computation costs sharply rise when the

number of dimensions and the sizes of training sets become large. The use of MapReduce as a processing platform has been regarded as a practical solution to resolve such criticism. The MapReduce framework takes the input data, depending on the size, it automatically splits the input data into smaller manageable chunks. Each smaller chunk is processed by a map task (also often called a mapper interchangeably). The result of a mapper is summarized as key and value pairs. The output (e.g., values) with the same keys are shuffled and reduced by a reducer function.

3.3 Related Work

Performing k-NN to provide performance gain has been extensively studied in the literature [122]. Nevertheless, this work only focuses on the centralized and single-thread approach that is not applicable in many modern-day applications which requires a large input data for computation. In [122], the authors reported the nearest neighbour classification with generalization applied in a large dataset. The main purpose of generalizing exemplars, which merges data into hyper-rectangles, is to improve speed and accuracy but they do not mention how to handle anonymized data. A privacy preserving classification techniques has been proposed by [17] . However, the techniques they use focus on neural network as an underlying algorithm then use homomorphic encryption [53] as a data anonymization technique. The direction they took is quite orthogonal to our work . In [138], the authors propose a new nearest neighbour approach using correlation analysis under a MapReduce framework on a Hadoop platform to address the difficult problem of real-time prediction with very large training datasets. However, using their approach, the performance of an algorithm can be seriously affected if the size of the training samples becomes extremely large. For many modern day uses of k-NN, the computational and the storage issues has become a critical problem [66, 159]. This is because the new applications of k-NN requires a rather large storage device to contain the whole training set as well as a large computation support in the classification stage. Airavat [110] proposed a framework for MapReduce by defining mandatory access control (MAC) with DP on a secure operating system SELinux. Airavat however describes a data anonymization via DP only with a very strict sensitivity pre-defined value which is only applicable to a specific case of applications where the distribution of the input data and types of operations performed on that data is pre-defined.

3.4 Data Anonymization

In this section, we discuss the details of K-anonymity on k-NN classifier for MapReduce operations. Our implementation is done based on the privacy-preserving platform we proposed previously [23]. The privacy preserving mechanism receives the user input data and defines a privacy protection mechanism, in our case K-anonymity. In the algorithm

Table 3.1: Original Adult Dataset

Age	Workclass**	Fnlwgt	Edu	Edu-num	Marital-status	Occupation	Relationship	Race	Sex	Capital-gain	Capital-loss	Hours-per-week	Native-country	Income
66*	Private	142624	Assoc-acdm	12	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	5556	0	40	Yugoslavia	>50K
55	Private	160631	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	4508	0	8	Yugoslavia	<=50K
53	Private	153064	5th-6th	3	Married-civ-spouse	Exec-managerial	Husband	White	Male	7688	0	10	Yugoslavia	>50K
51	Private	179479	HS-grad	9	Widowed	Exec-managerial	Not-in-family	White	Female	3325	0	40	Yugoslavia	<=50K
51	Federal-gov	223206	Doctorate	16	Married-civ-spouse	Prof-specialty	Husband	Asian-Pac-Islander	Male	15024	0	40	Vietnam	>50K

*a light grey represents an example of a tuple
** a dark grey shade represents an example of a feature

implementation layer, we choose k-NN as a classifier, transforms the original data into an anonymized equivalent still retaining the content value so that further analytics can be performed on the anonymized data. We measure the classification error on the privacy and utility measurement layer to understand the privacy and utility trade-off between the original data and the anonymized data.

3.4.1 Dataset and Pre-processing

We use the Adult dataset [15] to demonstrate our study. The dataset consists of personal information records extracted from the US census database. We use the dataset for a classification prediction as whether a given person has a potential to earn an annual income over or under $50,000. The original Adult dataset has six continuous and eight categorical features as seen in Table 3.1.

The k-NN algorithm often processes both categorical features and continuous features [38]. To overcome the difficulty of having to process the string data often found in categorical features, many implementations of k-NN often require the conversion of the categorical features to discrete numerical features. We adopt conversion from work of [67]., which utilizes unique numerical labels to convert each categorical value into its numerical counterparts. Using this technique, we transform eight categorical features (workclass, edu, marital-status, occupation, relationship, race, sex, native-country) into numerical features. For example, instead of using a country name such as Cambodia, Canada, China a numeric value is used such as 1 to represent the country Cambodia, 2 to as Canada so on. Table 3.2 represents the Adult dataset after the conversation of the categorical values.

3.4.2 k-NN Implementation on MapReduce

This section defines the k-NN algorithm we implemented in MapReduce operations. Our implementation strategies were inspired by the work on [91]. We use the following k-NN

Table 3.2: Adult Dataset after categorical value conversation to numeric

Age	Workclass**	Fnlwgt	Edu	Edu-num	Marital-status	Occupation	Relationship	Race	Sex	Capital-gain	Capital-loss	Hours-per-week	Native-country	Income
66	3	142624	8	12	3	7	1	5	1	5556	0	40	35	>50K
55	3	160631	12	9	3	7	1	5	1	4508	0	8	35	<=50K
53	3	153064	5	3	3	4	1	5	1	7688	0	10	35	>50K
51	3	179479	12	9	7	4	2	5	2	3325	0	40	35	<=50K
51	1	223206	11	16	3	9	1	2	1	15024	0	40	34	>50K

*a light grey represents an example of a tuple
** a dark grey shade represents an example of a feature

algorithm to get classification errors before data was anonymized. The general processing of data for MapReduce operations follows.

- Reading data: Consider a training dataset TR_s and a test dataset TS_s, they are formed by a number of records m-th (in TR_s) and t-th (in TS_s) respectively. Each training sample ST_r(line) is read and split as a tuple $(Tp_1, Tp_2, \ldots, Tp_D, W_c)$, where Tp_E represent E-th feature in p-th tuple, and ST_r belongs to a class W_c, for given $T_p^{W_c}$ and D diminutions.

- k-NN training: In order to train the k-NN algorithm, the training dataset TR_s should contain the value of W_c while it is unknown for the test dataset TS_s. For each test sample ST_r contained in the TS_s test dataset, the k-NN model looks for records whose distance proximity is smallest (i.e., indicating the records are similar) in the TR_s set. To do this, it computes the Euclidean Distances (ED) between TS_s and all ST_r of TR_s (i.e., for each sample of test dataset with all the sample of train dataset). Whereas the k-nearest neighbours' samples $(NB_1, NB_2, ..., NB_k)$ are obtained by ranking the training samples according to the computed distance.

- Alignment with Mapper operations: To apply this in the MapReduce model, we first organize a mapper to compute the classes W_c from the distance to the k- nearest neighbours for each test and training data.

- Alignment with Reducer operations: The reduce function is responsible for processing the Euclidean Distance (ED) of the k- nearest neighbours from each map and creates a list of k- nearest neighbours by taking those with minimum distance. Reducer shuffles the distances and examines for majority voting, then to assign the class for TS_s.

k-NN mapper and k-NN reducer are described in more detail as follows:

k-NN Mapper: In our implementation of k-NN for MapReduce, we represent our training set as TR_s and test dataset TS_s, both with a random number of records store in Hadoop Distributed File System (HDFS) as single file. The first step Mapper accesses the input file from the HDFS and disjoint TR_s into given number subsets. The training set TR_s is split into tuples containing the attributes (also known as features)

$(test_{tp_1}, test_{tp_2}, \ldots, test_{tp_D}, W_c)$, where, each $test_{tp}$ represent one feature of adult dataset and W_c represent as an income class (the feature to be classified). Suppose we have mappers from 1 to n, for each of the mapper task, it will create TR_{sj} from $1 \leq j \leq n$, which represent the training set sample ST_r. It should be noted that partitions of given processes are sequentially executed, for example, $mapper_j$ corresponds to j data chunk. In other words, each mapper will have its corresponding TR_{sj} and a class label W_c for every k- nearest neighbours. Each training record is divided into a subset of TR_s in order to compare each subset with its TS_s to find out a distance DC. The other small subsets are obtained based on k (degree of neighbours) and number of records in TS_s. Distances are stored in the distance matrix DC_j pairs as "(class, distance)" which can be represent as (W_c, k-distance) with dimension $k.m$ (i.e., DC_j compute all the distances for each tuple of TR_s with all element of TS_s). Each Row i will have W_c (classifier value) and k-nearest distance of class. The row i will repeat till t for each ST_s. After mapper completes its process, it stores the (key, value) pairs as ($(Mapper_{ID}, W_c), DC_j$), where $Mapper_{ID}$ is used to identify the mapper in single reducer. The complete pseudo-code for the k-NN mapper is described in the following Algorithm 3.

Algorithm 3: k-NN Mapper

Input: k value, TSs.
Output: key as mapper identifier $Mapper_{ID}$ and class W_c $value$ as DC_j
1: Create TR_{sj} with the instances of split j.
2: **for** i =0 **to** i<$size(TS_s)$ **do**
3: Compute k-NN $(ST_s, i, \mathrm{TR}_{sj}, k)$
4: **for** $m = 0$ **to** $m < k$ **do**
5: $DC_j(i, m) = < \mathrm{W}_c(NB_m), \mathrm{ST}_s(NB_m) > i$
6: **end for**
7: **end for**
8: $key = Mapper_{ID}, \mathrm{W}_c$
9: return (<key, DC_j >)

k-NN Reducer: Reducer is responsible for selecting most relevant neighbours, examines their classes and finds optimal classes for tuples in TS_s . The reducer phase can be divided into following four steps:

- The setup step reads and allocates the distance matrix DC_{reduce} of the fix size of (TS_s ,k-neighbours). DC_{reduce} value is assigned once a mapper completes DC_j and sends the data to reducer. Keys from the mapper is separated as $Mapper_{ID}$ and W_c. The size of the distance matrix is initialized with the total number of TS_s . Once the setup is done, it moves to the next reduce step.

- The reduce merge two sorted lists (i.e., one list containing the distances calculated with class and the other list contains the distances calculated with its neighbours).

Thus, for every , every distance is compared to its neighbours one at a time starting from the nearest one and sorted according to distances.

- The third step is clean-up. The clean-up process receives the list of neighbours for all TS_s as (class, distance) in the form of DC_{reduce} for majority voting in order to identify the predicted classes W_{cp} for TS_s . After the clean-up, the key value pair are redefined as TS_s classes W_{cp} and TR_s classes W_c.

- The comparison of the classifiers between these two classes are done in the classification error step to compute the error rate of k-NN for each k values. Following is the pseudocode that we use in this study for the k-NN reducer.

Algorithm 4: k-NN Reducer

Input: $Size$ of TS_s, k and DC_j.
Output: actual class W_c and predicated class W_{cp} as key and $Error\_rate$ as value

```
1:  for i = 0 to i < size(TS_s) do                          //setup
2:      cont = 0
3:      for m = 0 to k do                                   //reduce
4:          if DC_j(i, cont). ST_s< DC_reduce (i, m). ST_s then
5:          DC_reduce (i, m) = DC_j(i, cont).
6:          cont + +
7:          end if
8:      end for
9:  end for
10:     Error_rate =0, TP= 0, TN = 0
11: for l = 0 to l < size(TS_s) do                          // CleanUp
12:     PredClassl = MajorityVoting(Classes(DC_reduce ))
13:     key = W_cp , W_c
14:         If (W_cp == W_c) then
15:         TP ++
16:             else
17:             TN++
18:         end if
19:     Error_rate =(TP − TN /TP)                           // classification error
20: end for
21: return (<Error_rate >)
```

3.4.3 K-anonymity with k -NN in MapReduce

- In this section, we illustrate how we anonymize the original input data using K-anonymity technique. We run k-NN classifier on the anonymized dataset and get classification error. We compare the classifications errors obtained from a non-anonymized dataset as well as an anonymized dataset to understand whether there has been any impact on classification error. For this task, we extend the mapper operation in the previous section and produce multiple sets of anonymized datasets.

- The first step is to read TS_s into tuples containing the attributes (also known as features) $(test_{tp_1}, test_{tp_2}, \ldots, test_{tp_D}, W_c)$, where, each $test_{tp}$ represent one feature of adult dataset and W_c represent as an income class (the feature to be classified). The second step is to anonymize the number of features (α), while $test_{tp_\alpha}$ denote the particular feature to be anonymized.
- Then the third step is to assign K group size (KG) where KG is the degree of anonymity (i.e., the number of records to hide in a crowd).
- The fourth step is to calculate an average value on each attribute of α QIDs which to be anonymized. Now α values are replaced by the average of each feature.
- In the last step, we replace the average value against each value of KG in continuous features while the average value is used to find more generalized categorical value for the categorical converted numerical features. The input test data now changes from TS_s to it anonymized counterpart TS_{sa}.

The pseudo-code for this description is in the following Algorithm 5.

Algorithm 5: k-NN Mapper

INPUT: K group size as KG, $QIDs$ attribute as α, k-NN as k value, TRs and TSs.
OUTPUT: key as mapper identifier $Mapper_{ID}$ and class W_c $value$ as DC_j
Initialize all variables Avg=0, l=0
1: Read $test\_tp_\alpha$ as classifier from TSs
2: **for each** $i \in K$ **do**
3: KG = averages of all $K[l]$.
4: **for each** average **do**
5: $Noise[l] = Avg - KG[l]$;
6: $test\_tp_\alpha[i] = Noise[i]$
7: $test\_tp_\alpha$ store value on TS_{s1}
8: **end for each**
9: **if** (α is greater than initialized value) **then**
10: Goto Step 3 for every $test\_tp_\alpha$ on TSs
11: **end if**
12: **end for each**
13: Create TR_{sj} with the instances of split j.
14: **for** i =0 **to** i<$size(TS_{s1})$ **do**
15: Compute k-NN $(ST_s, i, \mathrm{TR}_{sj}, k)$
16: **for** $m = 0$ **to** $m < k$ **do**
17: $DC_j(i,m) =< \mathrm{W}_c(NB_m), \mathrm{ST}_s(NB_m) > i$
18: **end for**
19: **end for**
20: $key = Mapper_{ID}, \mathrm{W}_c$
21: return (<key, DC_j >)

Table 3.3: The sample of K-5 tuple with different number of column generalization

Age	Workclass**	Fnlwgt	Edu	Edu-num	Marital-status	Occupation	Relationship	Race	Sex	Capital-gain	Capital-loss	Hours-per-week	Native-country	Income
5 – anonymity with on 2 Features age and workclass														
55.2	2.6	142624	8	12	3	7	1	5	1	5556	0	40	35	>50K
55.2	2.6	160631	12	9	3	7	1	5	1	4508	0	8	35	<=50K
55.2	2.6	153064	5	3	3	4	1	5	1	7688	0	10	35	>50K
55.2	2.6	179479	12	9	7	4	2	5	2	3325	0	40	35	<=50K
55.2	2.6	223206	11	16	3	9	1	2	1	15024	0	40	34	>50K
5 – anonymity with on 4 Features age and workclass, fnlwgt and education														
55.2	2.6	171800.8	9.6	12	3	7	1	5	1	5556	0	40	35	>50K
55.2	2.6	171800.8	9.6	9	3	7	1	5	1	4508	0	8	35	<=50K
55.2	2.6	171800.8	9.6	3	3	4	1	5	1	7688	0	10	35	>50K
55.2	2.6	171800.8	9.6	9	7	4	2	5	2	3325	0	40	35	<=50K
55.2	2.6	171800.8	9.6	16	3	9	1	2	1	15024	0	40	34	>50K
5 – anonymity with on all Features														
55.2	2.6	171800.8	9.6	9.8	3.8	6.2	1.2	4.4	1.2	0	7220.2	27.6	34.8	>50K
55.2	2.6	171800.8	9.6	9.8	3.8	6.2	1.2	4.4	1.2	0	7220.2	27.6	34.8	<=50K
55.2	2.6	171800.8	9.6	9.8	3.8	6.2	1.2	4.4	1.2	0	7220.2	27.6	34.8	>50K
55.2	2.6	171800.8	9.6	9.8	3.8	6.2	1.2	4.4	1.2	0	7220.2	27.6	34.8	<=50K
55.2	2.6	171800.8	9.6	9.8	3.8	6.2	1.2	4.4	1.2	0	7220.2	27.6	34.8	>50K

In our study, we observe the effect of an anonymization in two different aspects: tuple-based vs feature-based generalization. We first examine the effect of anonymization by its usual tuple-based (i.e., making the number of records same), secondly, we examine the effect of anonymization by its feature-based (i.e., making the number of features across records same). For the former, we analyze 4 different tuple-based degree K=5,10,100,1000 for example K=5 indicates that there will be 5 records made same where K=10 indicates that there will be 10 records made same and so on. Simple represent no anonymization and transformation is applied on data. For the latter, we analyze 5 different feature-based degree A_x = 2, 4, 8, 12, all where x indicates the number of QIDs. From the feature-based generalization, $A2$ represents α=2, i.e., age and work-class are used as QIDs and generalized whereas $A4$ represents α =4, i.e., the four features age, work-class, fnlwgt, and education are used as QIDs. $A8$ and $A12$ represents in the similar fashion. We use the special notation AA to mean all features are used as QIDs and generalized accordingly. Table 3.3 represents the snippet of data anonymization on K-5 degree on different numbers of QIDs being generalized.

3.4.4 Differential Privacy DP with k-NN in MapReduce

For our study, we use Laplace distribution to calculate a noise. The noise is generated using Laplace distribution which we use for all features regardless whether it comes from categorical or numerical origins as all our features are represented as numerical values. Let a function f: $TS_s \rightarrow R$ over a dataset TS_s. By studying the distribution of the dataset TS_s the Laplace noise is calculated then added to each data uniformly.

$$M(TS_s) = f(TS_s) + Lap\frac{\Delta f}{\epsilon}, \tag{3.2}$$

Algorithm 6: Differential Privacy in k-NN Mapper

Input: k value, TRs , TSs, Privacy Budget as PB, Sensitivity Δf,
Output: key as mapper identifier $Mapper_{ID}$ and classes W_c, $value$ as neighbours for each k.
1: Create TR_{sj} with the instances of split j.
2: Calculate $Noise$ using $Lap(\Delta f/PB)$
3: **for** l= 0 **to** l< $size(TS_s - 1)$ **do**
4: $testtp[l] = testtp[l] + Noise$
5: Store $testtp[l]$ in TS_{s1}
6: **end for**
7: Repeat step 3 for every classifier $testtp$
8: **for** i =0 **to** i<$size(TS_{s1})$ **do**
9: Compute $k - \mathrm{NN}\ (ST_s, i, \mathrm{TR}_{sj}, k)$
10: **for** $m = 0$ **to** $m < k$ **do**
11: $DC_j(i,m) =< \mathrm{W}_c(nb_m), \mathrm{ST}_s(nb_m) > i$
12: **end for**
13: **end for**
14: $key = Mapper_{ID}, \mathrm{W}_c$
15: return (<key, DC_j >)

Table 3.4: Normalized features

age	workclass	fnlwgt	edu	edu-num	marital-status	occupation	relationship	race	Gender	capital-gain	capital-loss	hours-per-week	native-country	income
20.4	82.7	32.1	49.6	66.1	33.2	99.1	0.2	99.3	1.0	1.0	3.3	38.8	99.0	>50K
40.7	33.2	15.8	99.1	59.5	33.2	82.6	0.2	99.3	1.0	1.0	99.0	59.2	99.0	<=50K
40.7	49.7	30.6	74.3	85.9	33.2	24.8	0.2	99.3	1.0	1.0	14.8	49.0	99.0	>50K
47.5	16.7	29.1	55.8	79.3	33.2	66.1	0.2	99.3	100.0	1.0	7.5	38.8	99.0	<=50K
62.4	33.2	9.7	68.1	52.9	33.2	16.6	0.2	99.3	1.0	1.0	99.0	38.8	99.0	>50K

To apply DP in k-NN mapper, the first step is to read every features in the record TS_s. Then, we generate a noise using the above Laplace noise formula and add the noise to the features of every record in the entire data. The new values (i.e., the original features plus the noise) are stored in the anonymized dataset TS_{s1}. The pseudo-code applying DP for k-NN mapper is defined in Algorithm 6.

While calculating noises, data that contains a high sensitivity distribution, which can be seen in the features such as fnlwgt and capi-tal-loss in the Table 4.1d, needs to be normalised to have a smooth balance. To rectify the issue, we normalize TS_s and TR_s using the equation 3.3 as per the guideline of described by [33].

$$Xnorm_{a,b} = \lfloor \frac{(b-a)(X - Xmin)}{Xmax - Xmin} \rfloor + 1. \tag{3.3}$$

where a and b are the range of values in each features. In our experiment, we used normalization that ranges from 1 to 100. Where X is the value of features. By applying normalization, we can get a global view of our dataset by knowing the upper and lower value for each feature. After normalising we can find appropriate Δf for complete the dataset more easily. Table3.4 illustrates the feature values after the normalization.

3.5 Experiments and Results

A set of experiments have been conducted on the Adult dataset to observe the k-NN based classification errors on data anonymized using K-anonymity. The experiment was performed on the single node cluster with the following specification: (1) the CPU model: Intel(R) Xeon(R) CPU E5-1650 v3, (2) the processing speed: @ 3.50GHz, (3) the number of core processors: 6, (4) the storage capacity: 4 Terabytes, and (5) the memory size 32 GB of RAM. We first run the experiment on both the training and test datasets on k-NN classifier without any anonymization then check the classification error.

3.5.1 Applying K-anonymity on k-NN classifier

Fig. 3.1 illustrates the result on the classification error studying from the feature-based anonymization. For example Fig. 3.1(a) shows the results of 5 records anonymized on 2 QIDs = age, workclass which is denoted as K-5-A2 while the results of 5 records anonymized on 4 QIDs = age, workclass, fnlwgt, edu is denoted as K-5-A4. The same notation is used for other number of QIDs.

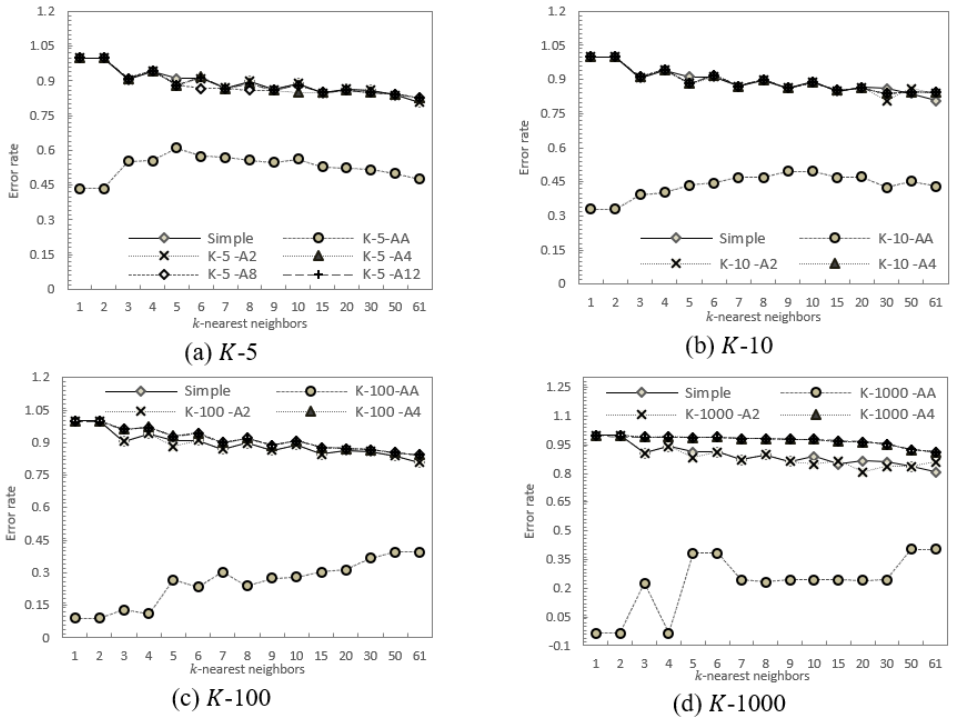


Figure 3.1: K-anonymity on varying degrees of anonymized feature sets

Here is the summary of our observations;

- The number of features being anonymized attributes to the decreasing accuracy (i.e., increasing classification error) as we see this in all graphs.

- As the number of k-nearest neighbours increases, more classification errors are generated. This is due to the increasing size of the sample being the subject of the classification, that is, there is increasing probability of producing an error as there are more data.
- There is a huge amount of classification errors when all features are anonymized in comparison to when there are at least a few features still not anonymized.
- The distribution of the data within a feature effects on the number of classification errors. If the distribution of the data is wide and if they are generalized, they tend to subject to more classification errors.
- With the increasing number of K degrees, the fluctuation of classification errors becomes unpredictable. For example, with K-5 and K-10, we observe a steady increasing or decreasing of classification errors in a smaller range scale which was between 45%-60% with K-5 while 33%-45% with K-10. In the meantime, the classification errors were sharply increased from 10% to 40% in K-100 while it was between 0% -45% with K-1000.

Fig. 3.2 illustrates the result on the classification error studying from the tuple-based–anonymization. For example, the 3.2(a) shows the results of 2 QIDs = age, work-class anonymized with different degree of K=5, 10, 100, 1000 . Here is the summary of our observations;

- There is more classification errors produced as the degree of K increases. It is easy to understand this pattern because simply more data means the increasing possibility with classification errors. This is observed in all graphs, irrelevant to the number of QIDs involved in the anonymization process.
- When only two QIDs were anonymized, as shown in Fig. 3.2(a), the effect of increasing K degree is negligent. As the number of QIDs increased to be anonymized, the scale of classification error range becomes wider. For example, in Fig. 3.2(a) where it is only two QIDs anonymized, there is almost no difference in classification errors among K-degrees. However, in Fig. 3.2(e) where all QIDs were anonymized, K-5 classification errors stay around 50%, K-10 classification errors stay around 30% whereas K-100 stays around 10%.
- The utility of anonymized data is higher with a fewer QIDs regardless K as we do not see much difference in the classification errors between non-anonymized data and anonymized data.

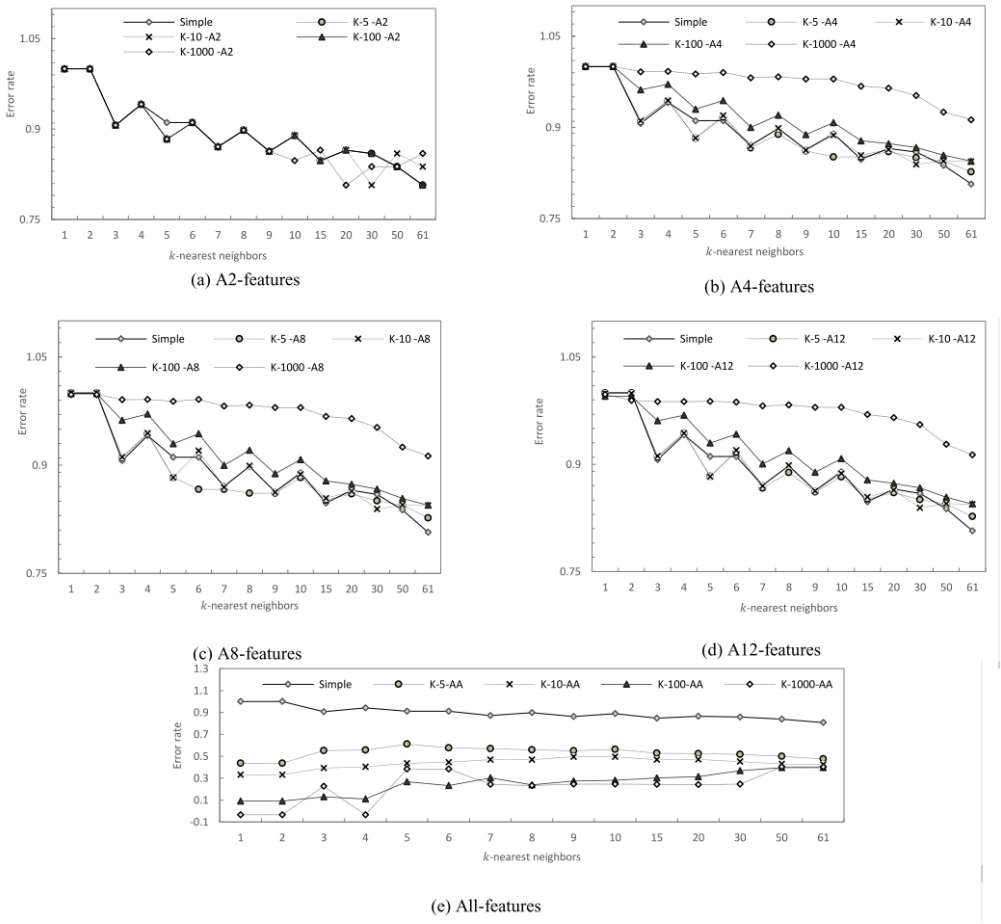


Figure 3.2: QIDs on varying degrees of K-anonymity

3.5.2 Applying Differential Privacy (DP) on k-NN classifier

In this section, we examine the effects of classification errors when DP is used as an anonymization technique. In our study, we applied DP on two slightly different versions of input data: (1) the first version of the data contains normalized values of pre-processed data where each feature in the data always ranges between 1 and 100, (2) the second version of the data contains only pre-processed Adult dataset without normalization applied.

Fig. 3.3 demonstrates the result of DP based anonymization on normalized data. The normalization was applied due to calculating more accurate global sensitivity which basis on inspecting on the distribution of data in the entire dataset. In this case, as we know the distribution of data, which is the range between 1 and 100 after normalisation, we used the sensitivity value as 1. We used varying degree of privacy budget $\epsilon = 0.001, 0.01, 0.1,$

1. In the figure, simple indicates k-NN classification errors obtained on non-anonymized and non-normalised dataset. Simple_normally indicates k-NN classification errors on non-anonymized and normalised dataset. ϵ =0.001 indicates the classification errors obtained on DP anonymized data with privacy budget of 0.001. The rest notations, with ϵ =0.01, ϵ =0.1, and ϵ =1, follows the same pattern. Here are a few observations we noticed:

- Clearly, non-anonymized data has a fewer classification error.
- Normalized non-anonymized data has more classification errors it is because normalization can be seen as anonymization itself as data has changed from its original form thus more errors introduced.
- The change of classification errors is negligible among DP anonymized data with different scale of privacy budget. It is because the distribution of data is smooth therefore the level of noise introduce by different privacy budgets are similar.
- There is steady increase of classification errors as the sample size of the data increase with the degree of k- nearest neighbour increases.

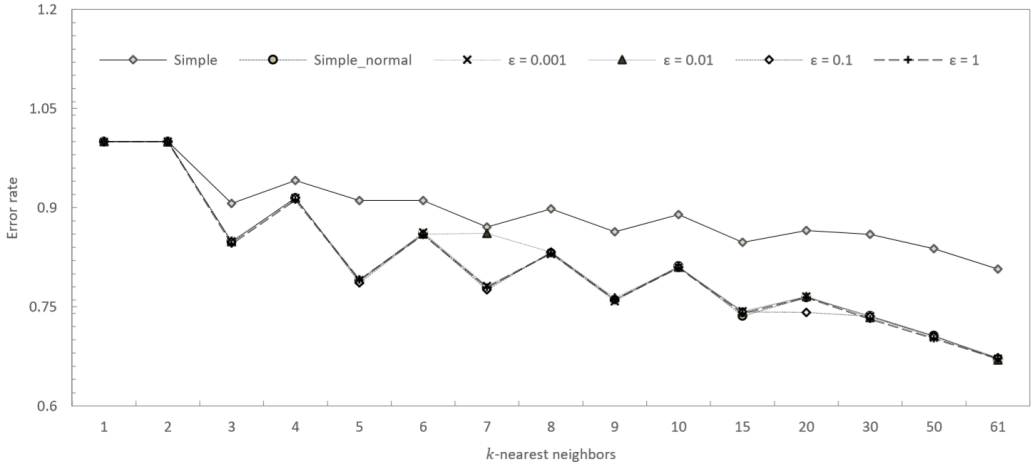


Figure 3.3: Differential Privacy on normalized Adult dataset

Fig. 3.4 demonstrates the result of DP based anonymization on non-normalised input dataset. In this figure, simple indicates k-NN classification errors obtained on non-anonymized dataset while the rest are anonymized with varying degree of privacy budget ϵ = 0.001, 0.01, 0.1, 1. As this is not normalized and we can't use the sensitivity value = 1, it is necessary to find a right level of sensitivity value that is more applicable for our input data. To do this, we calculated the average value of all attributes of the entire dataset and used it as a new sensitivity value. Our observation follows:

- Compared to the normalized counterpart in the above, the effects on the classification error on different privacy budgets are more visible. For example, non-anonymized dataset has the smallest classification error followed by ϵ =0.001 and ϵ =0.1. The most classification errors were observed in the ϵ =0.01 and ϵ =1.
- Privacy budgets ϵ =0.001 and ϵ =0.1 provides a higher utility that is almost close to the original dataset especially on the smaller k-nearest neighbour degrees (e.g., from degree 1 to 8).
- Similar to what we observed with the normalized dataset, it is possible to find optimal privacy budgets and sensitivity values that can produce better utilize as close to the original data for a specific k-nearest neighbours as seen with ϵ =0.001 and k neighbours degree=7.

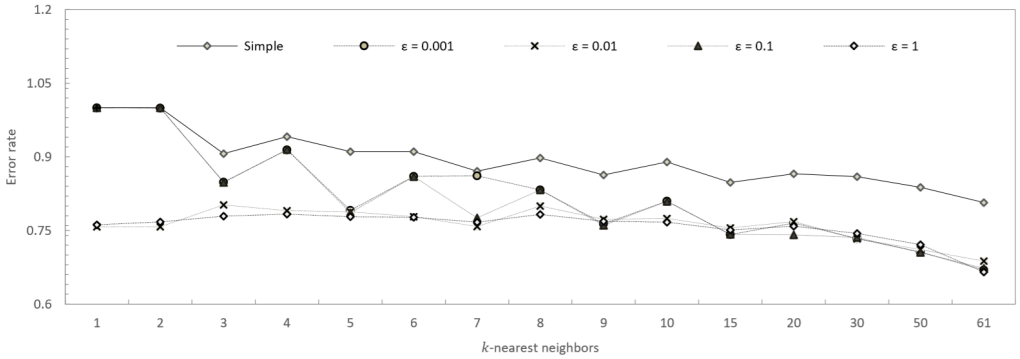


Figure 3.4: Differential Privacy on Adult dataset

3.6 Conclusions and Future Work

This research work is an extension from our previous work [23] where we focus running a classification algorithm on the anonymized dataset running a MapReduce platform. In this research we used k-NN as a classifier on anonymized data. We used the measurement of classification errors to observe the effects between privacy verses utility trade-offs when different sets of data were anonymized using multiple privacy parameters of K-anonymity. We used two different approaches; anonymizing the data based on (1) tuples and (2) features. As expected, the number of k-nearest neighbour has the close relationship with classification errors introduced. More data in a dataset produced higher probability of classification errors. We also observed that the distribution of the data within a feature for given dataset affects quite significantly on classification error.

For differential privacy-based data anonymization, we used an input dataset in two different ways to see the effect of data distribution among data: one normalized and the

other not. Normalized data has more classification errors. It is because normalization can be seen as anonymization itself. There is steady increase in classification errors as the sample size of the data increases with the degree of k- nearest neighbour increases. In this study, we observed that it is possible to find more optimal privacy budgets and sensitivity values that can produce better utility as close to the original data for a specific k-nearest neighbour.

In our future work, we plan to run our experiments in multiple node cluster which may need modification in the algorithms we used in this study. We also plan to make more close observations on the classification errors on different parameters on K-anonymity and differential privacy such as finding the most optimal point for privacy and utility trade off.

Chapter 4

Scalable and High Performance RDD-Based Data Anonymization for Spark

Summary

Recent studies in data anonymization techniques have primarily focused on MapReduce. However, these existing MapReduce based approaches often suffer from many performance overheads due to their inappropriate use of data allocation, expensive disk I/O access and network transfer, and no support for iterative tasks. We propose "SparkDA" which is a novel anonymization technique that is designed to take the full advantage of Spark platform to generate privacy-preserving anonymized dataset in the most efficient way possible. Our proposal offers a better partition control, in-memory operation and cache management for iterative operations that are heavily utilised for data anonymization processing. Our proposal is based on Spark's Resilient Distributed Dataset (RDD) with two critical operations of RDD, such as FlatMapRDD and ReduceByKeyRDD, respectively. The experimental results demonstrate that our proposal outperforms the existing approaches in terms of performance and scalability while maintaining high data privacy and utility levels. This illustrates that our proposal is capable to be used in a wider big data applications that demands privacy.

4.1 Introduction

The rapid growth of data from many domains (e.g., social media, smartphones, IoT etc.) has brought in a new era where extracting potential information using data analytic and data mining has become a top business priority to many organizations. Such practices, however, have also brought up data privacy concern in the absence of appropriate data protection mechanisms.

Data anonymization approaches are used to conceal private information in such a way where identifiable (sensitive) information is buried among non-identifiable groups [18,50]. Many different data anonymization algorithms have been proposed for the purpose including K-anonymization [127], l-diversity [88], t-closeness [86] and others [75, 123].

The recent growth of big data has created a high demand for distributed processing platforms that are equipped with a core set of features, for example, scalable processing units, large execution engines, and high capacity storage. Many existing anonymization methods used to run on a single machine have been redesigned to work with these new platforms (e.g., MapReduce) as the size of the input data increases massively [23, 24, 40].

In addition, many existing researches show that data anonymization methods implemented on MapReduce platform often have performance bottlenecks because underlying platform does not have appropriate supports for many core anonymizations tasks. These includes; MapReduce does not have a support for allocating data across partitions in different nodes in a balanced fashion which increases network overhead, doesn't support cache operation for saving the data produced while a task is still processing (e.g., intermediate data) which results in the intermediate data often being stored in the disk and fetched whenever it is needed [151], and does not have a support for iterative tasks which increases many performance overheads in terms of memory and network management [58]. More details of the issues associated with MapReduce are discussed in Section 4.3.1.

To address the limitations of MapReduce, Reference [144] proposed a new platform named Spark and since been hailed as the next generation of distributed processing platform. Spark has extended its scalability aspect in addition to offering a new set of advanced features more suited for the algorithms dealing with many different types of big data operations [116]. With the surge in the population of Spark and shift from MapReduce approach, many Spark-based data anonymization techniques have been proposed [9–11, 107, 119]. However, these existing proposals often tend to focus their efforts on improving and readdressing the scalability aspects to be more suited for Spark instead of investigating the suitability of Spark as a platform of choice for data anonymization techniques.

This is an extension of the earlier version which we presented in Reference [19]. The focus of the original paper was to present the details of a novel data anonymization approach based on Spark to take the full use of the advanced features offered by Spark while this extension offers an extensive evaluation for the suitability of our proposal for data anonymization techniques. By adapting and improving the advanced features of Spark, our approach effectively addresses many shortcomings of existing MapReduce based data anonymization approaches to resolve the overheads associated with expensive disk I/O, network and iteration tasks. We have extended our earlier version in several aspects. The new contributions of this paper are listed as follows:

- We provide clearer example of a general approach involved in a basic data anonymization technique with the addition of a flowchart to assist the understanding of the main tasks involved in such a technique. In addition, an additional mapping table is provided to further illustrate the relationship between the symbols and notations we use and database concept.
- We provide more detailed description of two critical RDDs involved in our proposal, FlatMapRDD and ReduceKeyByRDD respectively. These are designed to provide a better partition management, in-memory access for various data produced during anonymization process, and an effective cache management. We provide a better description as how our RDD-based approach can effectively reduce the significant overhead associated with MapReduce counterparts.
- We provide a new performance comparison between our proposal and the most up to date existing K anonymity based approaches and evaluates that our proposal offers a very competitive performance advantage.
- In addition to additional utility measurement matrices for Discenibility Metric (DM) and Minimal Distortion (MD), we provide a new set of privacy measurement matrices, such as Kullback-Leibler-Divergence (KLD) and Information Entropy (I_E), to extensively investigate the privacy and utility trade-offs of our proposal.
- We also provide the insights of a new set of performances associated with different memory management strategies offered by Spark. We discover that side-effect can occur when there are too excessive demands for memory access.

The chapter is structured as follows. In Section 4.2, we provide the recent related studies, while in Section 4.3, we provide the issues associated with MapReduce approach along with the description of a basic data anonymization technique as backgrounds. In Section 4.4, we discuss the details of our proposal along with main algorithms involved in our RDDs. Section 4.5 describes the details of the number of privacy and utility matrices we utilise and how we use them in the context of our proposal. In Section 4.6, we discuss the results of our experiments and the key findings. Section 4.7 provides the conclusion and planned future work.

4.2 Related Work

To address the overheads associated with MapReduce, a number of Spark based approaches have been proposed in recent years. Authors in [28] proposed the INCOGNITO framework for full-domain generalization using Spark RDDs. Though their experiential results illustrate the improvement in scalability and execution efficiency, their proposal does not provide any insights of privacy and utility trade-offs. Anonylitics [107] utilized Spark's

default iteration support to implement data anonymization. However, their approach does not address the potential memory exhaustion unable to accommodate increasing number of intermediate data produced as the number of iteration increases. PRIMA [11] proposes an anonymization strategy for Mondrian algorithm with Optimal Lattice Anonymization (OLA) which is used to define the utility and generalization level rules in order to limit the data utility loss. In [97], authors propose a distributed Mondrian approach by splitting the input data to the partitions allocated to each node of cluster by using Spark k-mean. A series of Spark jobs runs on each cluster node to produce anonymized results. These anonymized results are then merged together later by another cluster node.

The study that is most close to ours is that in [119] which provided a distributed Top-Down Specialization (TDS) algorithm to provide K-anonymity using Apache Spark. Rather, their solution focuses on addressing scalability and partition management which was originally proposed by [156]. They neither provide the details of the Spark feature they utilized nor any insights of privacy and utility trade-offs. Al-Zobbi et al. [10] proposed a sensitivity-based anonymization using user-defined function in Spark. The authors provide a strategy for reduced data transmission between memory and disk based on serialized data objects implemented with RDD and validate that a Spark-based approach can be many times faster than MapReduce counterparts such as [151].

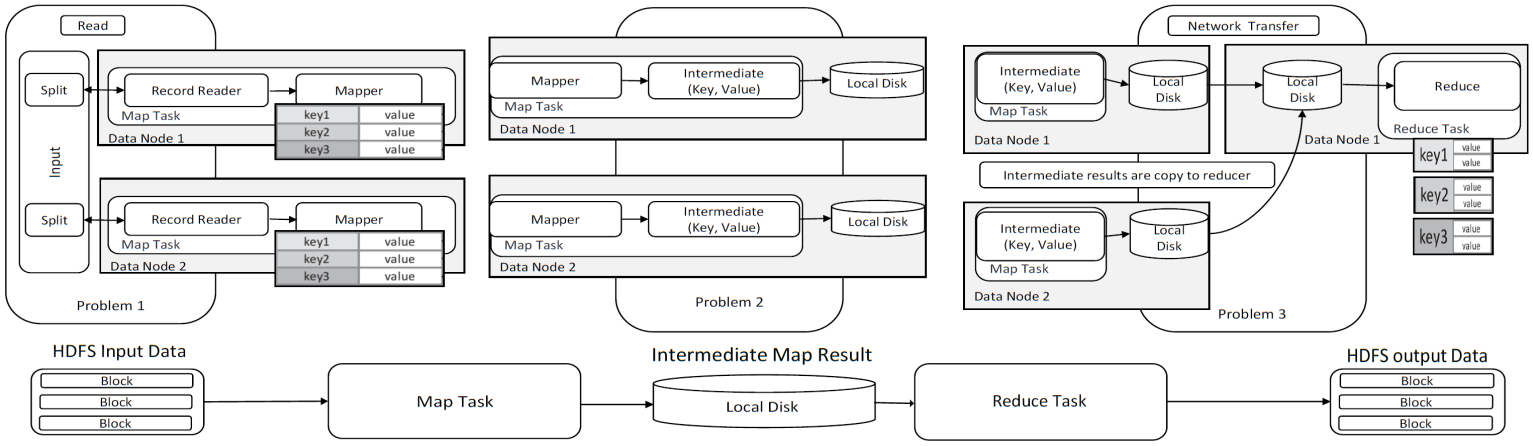
4.3 Background

We first provide the comparison of the difference and issues involved in MapReduce and Spark. This is followed by the description of the main tasks involved in a basic data anonymization strategy (e.g., Datafly [126]).

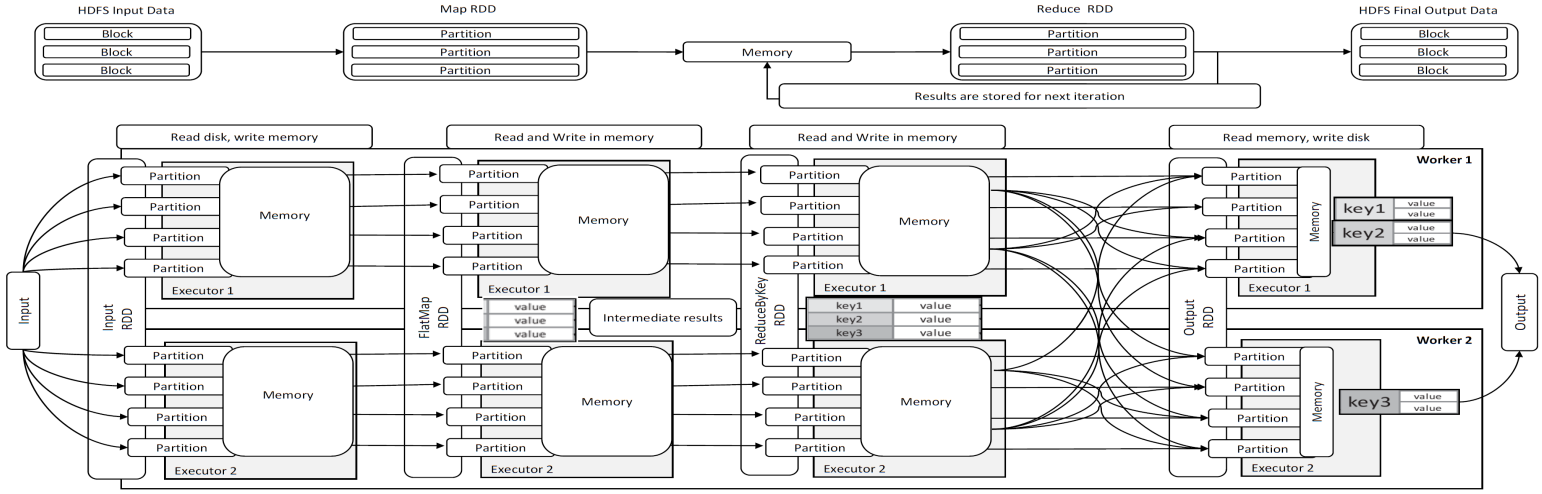
4.3.1 MapReduce vs Spark

For many years until Spark, Hadoop MapReduce [40] has been a widely used distributed processing platform for many big data applications. The fundamental building blocks of MapReduce are Map and Reduce. At start, MapReduce divides the (large) input data into a several smaller chucks. Each chunk of data (i.e., typically a collection of records) is mapped to a map across multiple mappers. The data contained in a mapper is assigned for a key-value combination. Each mapper process the data based on the key-value pair and the results, often called as intermediate data, is stored in the local disk where the mapper resides. Once the processing of all mappers are complete, a reducer reads the results from all mappers. Fig. 4.1(a) shows the full execution cycle of a MapReduce job and data movements involved at each phase. We argue that many performance overheads occur while MapReduce executes a job, especially in the following phases.

- Problem 1: One of the implications associated with MapReduce is with the creation of mappers where the size and number of mappers are decided without the consideration of the capability of each node. Once data is allocated to mappers, it is



(a) MapReduce Execution Cycle



(b) Spark Execution Cycle

Figure 4.1: Comparing the components and Data-flow in MapReduce and Spark structures. (a) MapReduce structures, (b) Spark structures.

not possible to re-allocate records across different mappers. This creates a several performance issues. Consider a case where a mapper is allocated with a larger set of records compare to other mappers. The execution for this mapper requires the use of the majority of the memory at the local disk while other mappers attached to the same node which shares the same local disk have to wait until the memory is freed. This can cause creating a long execution queue. Consequently, it can also cause a massive delay in the reducer in which waits for a long time until the mapper with the larger dataset completes despite all other mappers have already completed and their results are available much earlier. This problem is demonstrated as "Problem 1" in Fig. 4.1.

- Problem 2: Each mapper writes the results of the processing at the mapper in the local disk as intermediate data. The reducer requires accessing the intermediate data for further processing. This can cause the increase of expensive disk I/O by the mappers and the reducer when the number of intermediate data increases. This problem is demonstrated as "Problem 2" in Fig. 4.1.

- Problem 3: In MapReduce, a reducer processing the results of many mappers may reside in a separate network node. In this case, the results of mappers (i.e., intermediate data) requires to first read from the disk associated with the mapper, transferred across the network, and finally saved in the reducer's disk. As number of mappers increase, this can cause a significant network bottleneck especially if a particular network is slow or unavailable. This problem is demonstrated as "Problem 3" in Fig. 4.1.

- Problem 4: In case of a task with iterative nature, the result is first written in the local disk. If this result needs to be used again in the subsequent iteration, the mapper needs to access the disk again for each iteration. This architectural design is not only ineffective but also results in a tremendous performance bottleneck as it would cause a severe execution queue. To avoid the queue, the developer of MapReduce requires creating a series of sequential MapReduce jobs for the mappers manually. Even with this choice, it is often necessary that each iteration is waited for the completion (due to the issue discussed in the Problem 1).

Spark utilises Resilient Distributed Datasets (RDDs) as the building block to process Spark jobs. RDDs hold immutable collection of records which are partitioned and can be processed separately in parallel. Similar to MapReduce, input data is spilt as several smaller blocks. Each block then can be further divided into several partitions. An input RDD is created to hold all the partitions in the beginning. It then assigns partitions in the manner accounting for the processing capability at each worker's node to have the optimal number of partitions that can work most effective at each node. This new capability of Spark can reduce the issue associated with the Problem 1 we discussed earlier.

Once the initial partition allocation is complete, more RDDs are created to process the data contained in each partition – this is called a transformation in Spark. The intermediate data created by each RDD transformation is written in the memory and referenced as necessary. The memory accessibility can effectively reduce the performance overhead we discussed in the Problem 2 and 4.

In MapReduce jobs, the execution of each node happens as a separate unit of work. The result of each node, the collection of intermediate data, is not shared but being written off at each node due to the data locality principle of MapReduce. The only way to share the intermediate data with a reducer is via data transfer across networks. Spark offers the data sharing across different RDDs including the results produced by the previous stages and the intermediate data produced by different RDDs. This new feature of Spark can address the concerns we discussed in the Problem 3 and 4.

The execution flow of Spark is illustrated in Fig. 4.1(b) from data reads off the input data to the memory, processing data at different partitions, and then processing the partitions through RDD transformations.

4.3.2 Data Anonymization

Data anonymization refers to a process of transforming a set of original data into an anonymized data in such a way that uniquely identifiable attributes no longer present in the anonymized dataset while preserving statistical information about the original dataset. Two separate techniques are used for data transformation: generalization and suppression, respectively.

- Generalization involves with a process to replace the value of an attribute to a less specific value. Domain Generalization Hierarchy (DGH), which is typically defined and provided by a domain expert, is used to find the granularity for the generalization levels to be applied for each attribute.
- Similar to generalisation, suppression involves with a process replacing the original attribute to the value that does not release any statistical information about the attribute at all.

Fig. 4.2 demonstrates a generalization approach for applying generalization levels (GLs) defined in a DGH. For example, GL0 represents the first level of generalization while higher levels of generalizations are presented by GL2 and GL3. "*" is an example of suppression which appears in many attributes as the highest generalization level. Each "*" represents a numerical value of a generalization level, such as 114* represents GL1 while 11**, 1*** and * represents GL2, GL3, and GL4 respectively.

Though many variations of data anonymization methods have been proposed, our approach follows the one that is similar to Datafly [126]. The flow of Datafly algorithm

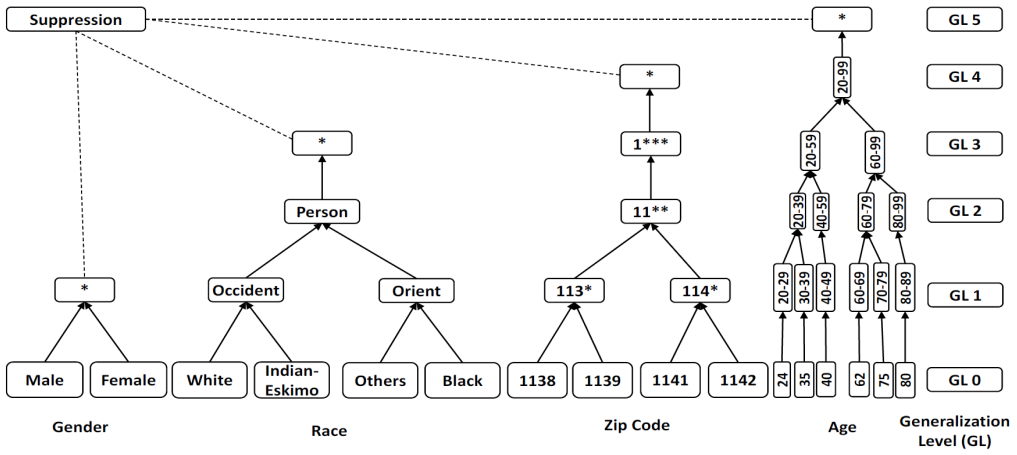


Figure 4.2: Examples of Generalization and Suppression for a Domain Generalization Hierarchies (DGH)

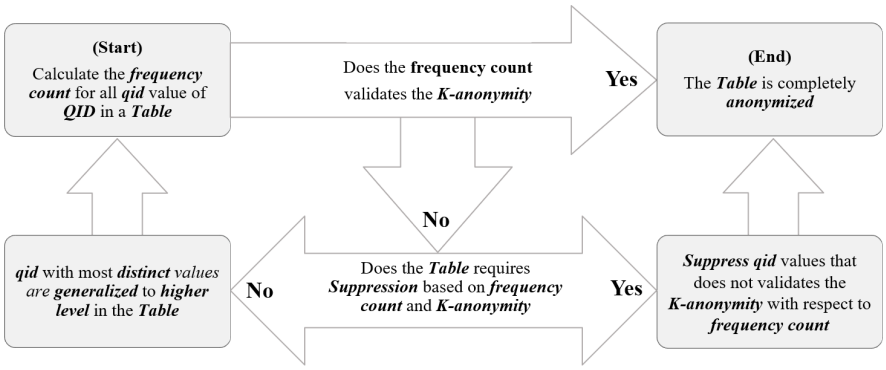


Figure 4.3: Datafly Algorithm

is depicted in Fig. 4.3. In this approach, data anonymization starts by counting the frequency, which represents the number of appearances given the record set, over the Quasi Identifiers Attributes (QID). The QID refers to a set of attributes that can uniquely distinguish an individual (e.g., age, date of birth, or address). Taking from the attribute with the most number of frequency count, the technique generalizes each attribute until K-anonymity constraint [127] is fully satisfied.

Table 4.1 illustrates the number of iterations in which a generalization is applied from the original data to a fully anonymized dataset. It starts with the original data depicted in Table 4.1(a). The original data is transformed based on the counting of the frequency of unique attributes and the frequency of unique tuples. Table 4.1(b) now contains the frequency counts. Starting from the attribute with the highest number of the frequency count, generalization based on DGH, an example shown in Fig. 4.2, is applied. For example, the attribute "Age" is first generalized because it has the highest number of

the frequency count at 6. Table 4.1(c) depicts a partially anonymized data. Note that a multiple level of generalizations can be performed at this stage as long as it doesn't violate the K-anonymity constraint. The final fully anonymized result is presented in Table 4.1(d) which meets the $K = 2$ constraint.

Table 4.1: Data Anonymization Steps

(a) Original Data

Age	Gender	Zip Code	Race	Disease
24	Female	1141	White	Fever
35	Female	1141	White	Fever
40	Male	1138	White	Back Pain
62	Female	1139	Black	Asthma
75	Male	1138	Black	Heart Attack
85	Male	1138	Black	Heart Attack

(b) Frequency counts

Age	Gender	Zip Code	Race	Frequency	Tuple
24	Female	1141	White	1	T1
35	Female	1141	White	1	T2
40	Male	1138	White	1	T3
62	Female	1139	Black	1	T4
75	Male	1138	Black	1	T5
85	Male	1138	Black	1	T6
6	**2**	**3**	**2**		

(c) Partially Anonymized Data

Age	Gender	Zip Code	Race	Frequency	Tuple
20-59	Female	1141	White	2	T1,T2
20-59	Male	1138	White	1	T3
60-99	Female	1139	Black	1	T4
60-99	Male	1138	Black	2	T5,T6
3	**2**	**3**	**2**		

(d) Fully Anonymized Data

Age	Gender	Zip Code	Race	Disease
20-59	Female	1141	White	Fever
20-59	Male	1138	White	Back Pain
60-99	Female	1139	Black	Asthma
60-99	Male	1138	Black	Heart Attack

4.4 SparkDA

In this section, we describe the details of our approach named SparkDA. We first provide the descriptions for the symbols and notations we used. Then, we describe our two RDDs, FlatMapRDD and RedueByKeyRDD, and the algorithms each of the RDDs executes.

4.4.1 Basic Symbols and Notations

The elements of the data across different scopes are outlined using the symbols and notations in Table 4.2. The mapping diagram of our proposed notations to a relational database concept is demonstrated in Fig. 4.4.

4.4.2 RDD-Based Data Anonymization

In our proposed approach, a data anonymization technique is implemented through the use of two Spark RDD transformations, FlatMapRDD and ReduceByKeyRDD, respectively.

4.4.2.1 FlatMap Transformation (FlatMapRDD)

The overall purpose of the FlatMapRDD is to compute for both the frequency of distinct attributes and the distinct tuples for all quasi-identifiable attributes. The frequency counts are then used to decide if further anonymization is necessary.

Table 4.2: Basic Symbols and Notations

Symbol	Definition
PT	A table (dataset) that contains records
RECORD(r)	A record contains a number of attributes, $RECORD \in PT$ and $RECORD = \{qid_1, qid_2, \ldots, qid_{attr}, sa\}$, where $qid_i, 1 \leq i \leq attr$, is the qid attribute and sa sensitive attribute
$attr$	Indicates a quasi-identifiable attribute
qid	A quasi-identifier attribute
QID	A set of attributes that belongs to the same qid
sa	Indicates a sensitive attribute
SA	Contains a set of attributes that belongs to the same sa
qid_{tuple}	Contains all qid(s) within a record $qid_{tuple} = \{qid_1, qid_2, \ldots, qid_{attr}\}$
QID_{Tuple}	Contains a set of qid_{tuple}, $QID_{Tuple} = \{qid_{tuple_1}, \ldots, qid_{tuple_{attr}}\}$
$freq(qid_{tuple})$	A set that contains a frequency associated to a qid_{tuple} for all qid_{tuple}(s) within a QID_{Tuple}
$freqSet$	A set that contains $freq(qid_{tuple})$ associated to a qid_{tuple}, $FreqSet = \{(qid_{tuple_1}, freq(qid_{tuple})_1), \ldots, (qid_{tuple_{attr}}, freq(qid_{tuple})_{attr})\}$
$dint_{qid}$-cnt	A number of occurrences for a distinct QID(s)in qid
$dint_{qid}$-$cntSet$	A set that contains $dint_{qid}$-cnt a associated to a QID for all qid(s) within a QID_{Tuple}, $dint_{qid}$-$cntSet = \{dint_{qid}\text{-}cnt_1, \ldots, dint_{qid}\text{-}cnt_{attr}\}$
DGH	A Domain Generalization Hierarchy
GL	Generalization Level of $QID \in DGH$
K	K defines the level of K-anonymization
EC	Finds the number of the same qid(s) within a QID for a given group based on K

The Algorithm 7 illustrates the working of the FlatMapRDD algorithm. The algorithm starts by loading the input data into QID_{Tuple}. At this initial stage, the QID_{Tuple} contains the original quasi-identifiable attributes.

The first part of the algorithm (depicted by step 2–8) executes to identify the frequency counts. To do this, it first measures the size of QID_{Tuple} to compute the total number of qid_{tuple} it contains (in step 3). The current qid_{tuple} is compared to the next qid_{tuple}. If a match is found between the two comparing qid_{tuple}(s), the frequency count is updated by adding the number 1. This is repeated for each and every qid_{tuple} within the QID_{Tuple}. However, the algorithm does not update frequency count if the qid_{tuple} and the subsequent qid_{tuple} values are different as this indicates two different records. When the iteration through QID_{Tuple} completes, the frequency counts for each unique tuple for all qid_{tuple}(s) is saved in the $FreqSet$ (seen in step 7). It should note that Spark sorts the qid_{tuple}(s)

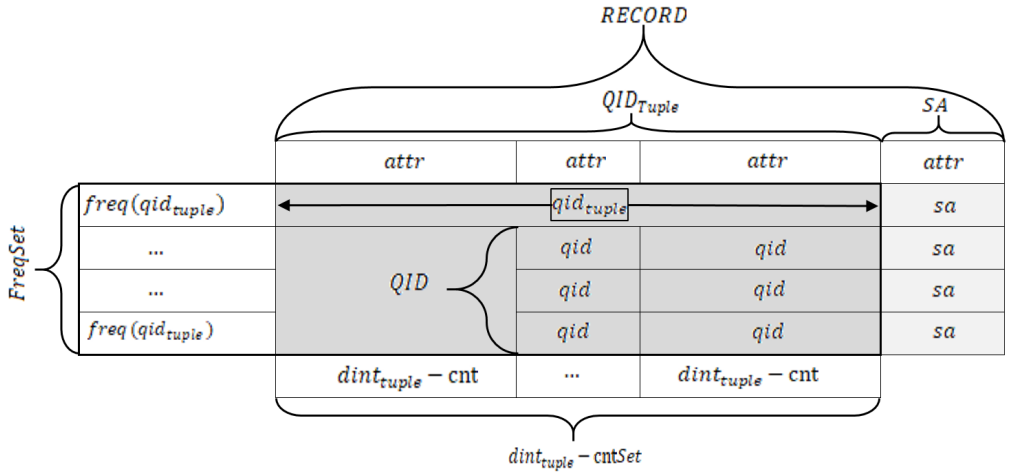


Figure 4.4: Notations Mapped for a Database Table

within the partition of each executing node and the frequency count of each qid_{tuple} is always equal to the number of respective qid_{tuple} appearing in the dataset as the total frequency count for all qid_{tuple}(s) represent the sum of records in the dataset.

The second part of the algorithm (depicted by step 9–22) runs to identify the count for the distinct attribute within a QID. To do this, it first measures the size of QID_{Tuple} to compute the total number of QID(s) it contains. Subsequently, the current qid is compared to the next qid. If a match is found between the two qid(s), the distinct qid count is updated by adding the number 1. This is repeated for each and every qid given the QID. When the iteration through QID(s) completes, the distinct counts for each unique attribute for all qid(s) is saved in the $dint_{qid}$-cnt (seen in step 22). The algorithm returns $FreqSet$ and $dint_{qid}$-$cntSet$ along with QID_{Tuple} to ReduceByKeyRDD.

4.4.2.2 ReduceByKey Transformation (ReduceByKeyRDD)

The overall aim of the ReduceByKeyRDD is to execute an RDD transformation by applying a generalization level using the information contained in $FreqSet$ and $dint_{qid}$-$cntSet$. The RDD transformation can be interpreted as the changes made to the original data in Table 4.1(a) until it reaches the results seen in Table 4.1(d), through Table 4.1(b) and Table 4.1(c). We introduce an "anonymization statue (represented by a variable = $anonymization_s$)" to keep track of whether a given QID_{Tuple}, which contains the lasted anonymization results, is fully anonymized or not and if a further anonymization processing is necessary. The Algorithm 8 illustrates the working of the ReduceByKeyRDD algorithm. To start the algorithm, the combination of (DGH, K) which contains the taxonomy tree and the K-anonymity constraint, is received via a broadcast mechanism which is sent by the driver node. DGH is further used to retrieve the generalization level (GL) for each quasi-identifiable attribute. This is described in step 3–4.

Algorithm 7: FlatMapRDD

```
   Input: QID_Tuple
   Output: FreqSet, dint_qid-cntSet
 1 begin
 2   freq(qid_tuple) = 1
 3   for i in Size(QID_Tuple) do
 4     if qid_tuple_i = qid_tuple_i+1 then
 5       freq(qid_tuple) ++
 6     end
 7     FreqSet+ = (qid_tuple, freq(qid_tuple))
 8   end
 9   dint_qid-cnt = 0
10   for i in Size(QID_Tuple) do
11     for j in Size(qid_tuple) do
12       QID_j = qid_tuple(i)(j)
13     end
14   end
15   for i in Size(QID) do
16     if qid_i = qid_(i+1) then
17       dint_qid-cnt_(i)
18     end
19     else
20       dint_qid-cnt_(i) ++
21     end
22     dint_qid-cntSet += dint_qid-cnt_(i)
23   end
24   return (FreqSet, dint_qid-cntSet)
25 end
```

The first part of the algorithm (depicted by steps 6–18) is operated to apply a single generalization level in all quasi-identifiable attribute sets. Applying a generalization level is repeated until the frequency counts (freq(qid_{tuple})) does not exceed the size of K and also does not exceed the maximum generalization level (MAX(GL_{qid})). The generalization is applied to attributes with the highest distinct attribute counts (MAX($dint_{qid}$-cnt)) to lower. The anonymization status is set to false while generalization level is being applied.

The second part of the algorithm (depicted by steps 21–26) is operated by applying suppression for all attributes for a given tuple which have violated the K-anonymity constraint to ensure no indistinguishable tuples exists. By now, all anonymization is complete, including the suppression, therefore the anonymization status is set to true. As seen in step 29, the anonymized results are sent back to the FlatMapRDD along with the anonymization status. Upon receiving updated QID_{Tuple} which now contains the anonymized data, the FlatMapRDD computes again for the frequency counts for the distinct tuples and the distinct attributes if only the anonymization status is still set to false.

Algorithm 8: ReduceByKeyRDD

Input: $FreqSet$, $dint_{qid}$-$cntSet$
Output: QID_{Tuple}, $anonymization_s$

```
1  begin
2      (DGH, K) ← broadcast(DGH, K)
3      GL_qid ← (DGH, K)
4      K ← (DGH, K)
5      anonymization_s = false
6      for i in Size(FreqSet) do
7          if dint_qid-cnt < K then
8              for j = 0 in Size(dint_qid-cntSet) do
9                  if MAX(dint_qid-cnt_j) < MAX(GL_qid) then
10                     UPDATE qid_(i)(j) with value of GL_qid_j + 1
11                 end
12                 else
13                     qid_(i)(j)
14                 end
15                 qid_tuple+ = qid_(i)(j)
16             end
17             QID_Tuple+ = qid_tuple
18             anonymization_s ← false
19         end
20         else
21             for j in Size(qid_tuple) do
22                 UPDATE qid_(i)(j) with "*"
23                 qid_tuple+ = qid_(i)(j)
24             end
25             QID_Tuple+ = qid_tuple
26             anonymization_s ← true
27         end
28     end
29     return (QID_Tuple, anonymization_s)
30 end
```

4.4.3 Overall SparkDA Scheme

In this section, we describe the overall process of our proposed approach that includes both the data anonymization process by two RDDs we described earlier and how these RDDs interact with other parts of the program.

The overall algorithm for our SparkDA is illustrated in Algorithm 9. The algorithm runs first by reading off user defined information such as K (i.e., K-anonymity constraint) and DGH (i.e., contains the definition of generalization hierarchy), as depicted in step 3-4. The K and DGH are used as global variables that are shard across all Spark worker nodes associated with processing RDDs. Spark supports broadcast mechanism to send the global variables across worker nodes.

The original data file from HDFS is read and saved into an InputRDD (step 1). The InputRDD pre-processes the input data in such a way that is easier to be processed by other RDDs. For example, the input data is divided into two different datasets, one set

contains all quasi-identifiable attributes (QID_{Tuple}-RDD) while the other set contains all sensitive attributes (SA-RDD) (step 6). We cache SA-RDD and QID_{Tuple}-RDD as they are used in many subsequent processing. At this stage, the anonymization status is set to false (step 5).

As depicted in steps 9-14, now two RDDs involved in data anonymization process, FlatMapRDD and ReduceByKeyRDD, executes interactively many times. The anonymization process completes when the fully anonymized dataset QID_{Tuple} is returned from ReduceByKeyRDD in which the anonymization status is set to true. The anonymized dataset, a generalized and distinct qid_{tuple} contained within QID_{Tuple}, is finally joined with corresponding SA-RDD (step 16).

Algorithm 9: SparkDA

Input: Dataset, K, DGH
Output: $Anonymized(RDD)$

1 **begin**
2 $InputRDD \leftarrow textFile(Dataset)$
3 $broadcast(DGH, K) \leftarrow broadcast(DGH)$
4 $broadcast(DGH, K) \leftarrow broadcast(K)$
5 $anonymization_s = false$
6 $SA\text{-}RDD, QID_{Tuple}\text{-}RDD \leftarrow InputRDD.filter(qid_{tuple}, sa)$
7 $SA\text{-}RDD_c \leftarrow SA\text{-}RDD.cache$
8 $QID_{Tuple_c} \leftarrow QID_{Tuple}\text{-}RDD.cache$
9 **while** $anonymization_s = false$ **do**
10 $Result\text{-}RDD(QID_{Tuple}, anonymization_s) \leftarrow QID_{Tuple}$.FlatMapRDD($QID_{Tuple}$) .ReduceByKeyRDD($dint_{qid}$-$cntSet$, $FreqSet$)
11 $QID_{Tuple}\text{-}RDD.cache \leftarrow filter.Result\text{-}RDD(QID_{Tuple}, anonymization_s)$
12 $QID_{Tuple_c} \leftarrow QID_{Tuple}\text{-}RDD.cache$
13 $anonymization_s \leftarrow filter.Result\text{-}RDD(QID_{Tuple}, anonymization_s)$
14 **end**
15 $Anonymized_{Tuple} \leftarrow filter.Result\text{-}RDD(QID_{Tuple}, anonymization_s)$
16 $Anonymized(RDD) \leftarrow Anonymized_{Tuple}.join(SA\text{-}RDD_c)$
17 return $Anonymized(RDD)$
18 **end**

The details of Spark execution cycle according to the overall SparkDA operations is depicted in Fig. 4.5.

4.5 Privacy vs. Utility Trade-Offs

We used the following privacy and utility metrics to validate and understand the trade-offs between these two. In the study of understanding the success of a data anonymization technique, a privacy level is measured by identifying the uniqueness of data. With that, a low privacy typically means that it is easy to identify an individual (an attribute, tuple or record) from a group (e.g., many records are unique) while a high privacy indicates that it is (more) difficult to uniquely identify an individual from a group (e.g., there are many records sharing the same values). A utility level is measured by calculating the

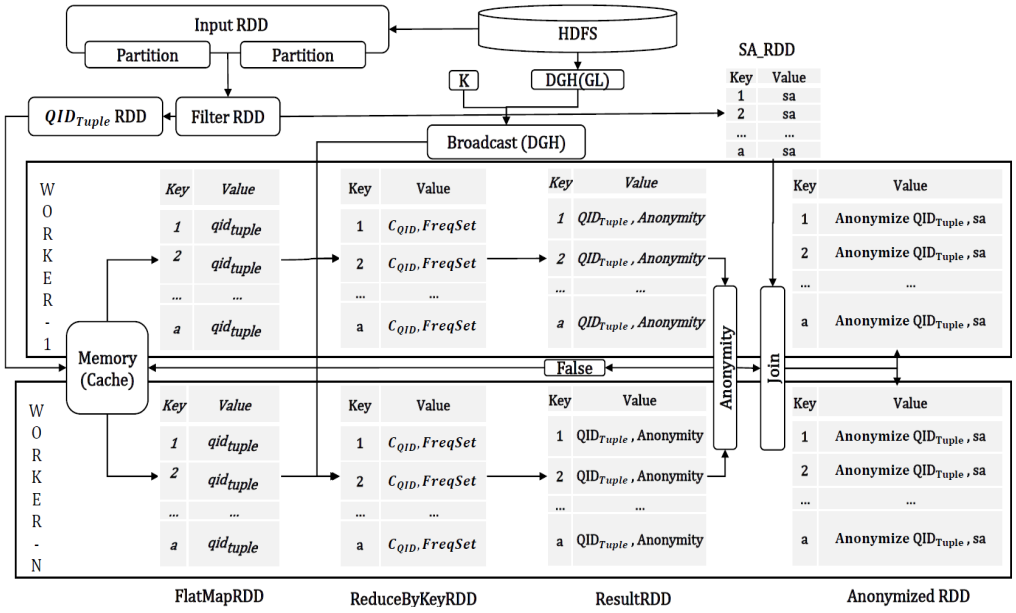


Figure 4.5: DataFlow in Spark

level of degradation in accuracy of value between the original value (i.e., baseline) and the anonymized value (i.e., sanitized).

4.5.1 Privacy Metrics

Privacy measure can ensure that the data is protected from any privacy attack after the data has been sanitized. This research work used following data utility matrices to validate our proposal.

4.5.1.1 Kullback-Leibler−Divergence (KLD)

KLD is utilized for understanding the likelihood of the presence of the original attribute in the anonymized attribute for each record [76]. For example, assume that the original attribute of the age 24 is anonymized into a range of 20–59. The KLD can measure what is the possibility of guessing the original age 24 from the range 20–59. Note that we use the term "likelihood" instead of "probability" to indicate that our calculation is done on the past event of the known outcomes (i.e., anonymized dataset). We measure KLD on the fully anonymized dataset by computing the followings: (1) calculating the likelihood of the presence of each attribute, (2) sums up all the value of (1) for each attribute within a record, then continues steps (1) and (2) for all records. Here, $P_{InputRDD}$ indicates the sum of the likelihood of the presence of the original attribute within the original data (at a record level). $P_{InputRDD}$ at this stage has a very high data utility and no privacy as there is no changes made. $P_{InputRDD(r)}$ indicates the sum of the likelihood of the presence

of the original attribute within the anonymized record. $P_{AnonymizedRDD}$ usually has lost some degree of data utility and has gained some degree of privacy because the data in this set has changed from the baseline after an anonymization technique is applied.

$$KLD = \sum_{r=1}^{n} P_{AnonymizedRDD}(r) \log \frac{P_{AnonymizedRDD}(r)}{P_{InputRDD}(r)}. \tag{4.1}$$

The KLD value increases from 0 which indicates both records between the original record and the anonymized record are the same. The increase of KLD value indicates the level of privacy assurance. With the lower value of KLD, it is easy to identify the original value from the matching anonymized value (i.e., low privacy).

4.5.1.2 Information Entropy (I_E)

The I_E is used to measure the degree of how uncertain it is to identify the original value from the anonymized value within the QID attributes [14]. The entropy value of I_E is 1 if all the *qid* attributes are identical in the anonymized dataset for the same QID. The $I_E(QID)$ value can be calculated by, (1) calculating the likelihood of the presence of the original attribute in a record, (2) computing the sum of the values of step (1) for each attribute in a record (denoted as $P_{AnonymizedRDD(qid)}$), (3) continuing the steps (1) and (2) for each QID, and (4) computing the sum of the value of step for all records. Note that if all attributes are changed between the original record and the anonymized record, the value of $P_{AnonymizedRDD}$ is 1.

$$I_E = -\sum_{qid=1}^{n} P_{AnonymizedRDD(qid)} \log P_{AnonymizedRDD(qid)}. \tag{4.2}$$

From Equation (—⋆—), we obtain $I_E(QID)$ for a single QID, however, we are interested in the I_E for the whole anonymizedRDD. Thus, we calculate the I_E for anonymizedRDD by taking the average of all QIDs. The entropy value of I_E is 0 if there are two identical records from the original dataset to the anonymized dataset for a matching equivalent class. The maximum value of I_E is achieved when the original record sets is completely different from the anonymized record sets for a given QID. Higher value of I_E represents more uncertainty (i.e., higher privacy).

4.5.2 Utility Metrics

Data utility can ensure that the data still contain enough information where data analytics can still find the relationships and correlations between data. This research work used following data utility matrices to validate our proposal.

4.5.2.1 Discernibility Metric (DM)

DM reports the data quality resulting from the degree of data degradation, as a result of data anonymization, of an individual tuple based on an equivalent class. Let EC be the set of equivalence classes of a K-anonymized dataset. EC_i is one of the equivalence classes of $\mid EC \mid$. The DM metric can be expressed more formally for AnonymizedRDD as follows:

$$DM_{score} = \sum_{EC_i \in AnonymizedRDD} \mid EC_i \mid^2, \tag{4.3}$$

where i represents a qid_{tuple} within an equivalent class. The data utility is associated with the DM score. If DM score is high, it means the data utility is low (i.e., the original qid_{tuple} has lost its original values) while the lower the DM score represents the data utility is high.

4.5.2.2 Average Equivalence Class Size Metric (C_{AVG})

C_{AVG} measures data utility of attributes by calculating the average size of the equivalence class. A higher data utility is typically achieved when the number of equivalence size is bigger because it is more difficult to distinguish an attribute when there are large number of attributes. Therefore, it is considered that the results of C_{AVG} scores are sensitive to the K group size [81]. We calculate C_{AVG} according to AnonymizedRDD as following.

$$C_{AVG} = \frac{|AnonymizedRDD|}{|EC|}/K, \tag{4.4}$$

where $|AnonymizedRDD|$ denotes the total number of records within the anonymized set while the total number of equivalence classes is denoted by $|EC|$.

4.5.2.3 Minimal Distortion (MD)

The MD measures data utility of every quasi-identifiable attribute (qid) in a tuple (qid_{tuple}). It defines data utility by comparing the rate where how many numbers of qid(s) in (qid_{tuple}) have been made to be indistinguishable. This is done by measuring the level of distortion on each qid in respect to a generalized level [83]. We calculate the distortion from the qid_{tuple} of AnonymizedRDD in comparison to InputRDD by using the following equation.

$$MD = \sum_{i=1}^{|D|} MD[Inputqid_{tuple-i}, Anonymizedqid_{tuple-i}], \tag{4.5}$$

where $|D|$ depicts the number of tuples in InputRDD. Equation (5) defines MD for complete dataset. The overall distortions between the anonymized dataset and the original dataset can be minimized by decreasing the K group size.

4.5.2.4 Precision Metric (PM)

As cited in Reference [126], PM calculates the least distorted combination of attribute and tuples from anonymized records. PM is typically considered to be sensitive to the GL. We define the equation for PM_{score} according to AnonymizedRDD as follows.

$$PM_{score} = 1 - \left(\frac{\sum_{qid=1}^{qid_{tuple}} \sum_{qid_{tuple=1}}^{QID_{Tuple}} \frac{GL}{|DGH_{qid_{tuple}}|}}{qid_{tuple}.QID_{Tuple}} \right), \tag{4.6}$$

where GL represents a generalization level (including suppression) which is defined in the DGH. The attributes associated with a higher generalization level tends to provide a better precision score than the attributes with a lower generalization level.

4.6 Experimental Results

This section first illustrates our experimental setups with the dataset and the system environment configurations. Then, we discuss the results of privacy and utility scores we obtained. The comprehensive experimental results of scalability, performance, and the impact of different cache management strategies of Spark follows.

4.6.1 Datasets

In our study, we used two datasets: US Census dataset (i.e., Adult dataset) [15] and Irish Census dataset [2]. We synthesized these datasets to increase the number of records to investigate different aspects of performance. We used "Benerator", which is a Java-based open-source tool, and the guideline from Reference [16] to generate the synthesized datasets. Table 4.3 illustrates the details of the both datasets including the quasi-identifiable attributes (QID), the number of district value, and generalization levels. The sensitive attributes are set to the "Salary" in the Adult dataset and the "Field of Study" in the Irish dataset.

Table 4.3: Datasets used in this study

(a) Adult dataset

QID	Distinct Value	*GL*
Age	74	4
Work Class	8	2
Education	16	4
Marital Status	7	3
Occupation	14	2
Gender	2	1

(b) Irish dataset

QID	Distinct Value	*GL*
Age	70	4
Economic Status	9	2
Education	10	4
Marital Status	7	3
Industrial Group	22	2
Gender	2	1

4.6.2 System Environment Configurations

Our experiments were run on two different platforms. The first sets of experiment were executed in a distributed processing platform environment using Spark while the other sets of experiment were executed on a standalone desktop. The latter was used to validate the comparability of data privacy and utility. The expectation was that the data privacy and utility scores should stay same between two sets of experiments.

We used Spark 2.1 where Yarn and Hadoop Distributed File System (HDFS) were configured using Apache Ambari. HDFS was used to distribute data across a NameNode (worked as a master node), a secondary NameNode, and six DataNodes (worked as worker nodes). 3GB memory was allocated to Yarn NodeManager while 1GB memory was configured for each of ResourceManager, Driver, and Executor memory. Table 4.4(a) shows the Spark and Hadoop Parameters while Table 4.4(b) provides the details of the Spark cluster and standalone desktop setups. Windows 10 was used as a standalone desktop. All experiments ran at least10 times and the average was used as to warrant the reliability and consistency of the results.

Table 4.4: Hardware and Cluster Parameters and Configuration

(a) Spark and Hadoop Parameters

Spark		Hadoop	
Resource Manager Memory	1 GB	NameNode	1
Driver Memory	1GB	DataNode	6
Executor Memory	1 GB	Block Replication	3
Driver Cores	1	Block Size	128MB
Executor Cores	1	HDFS Disk	18 TB

(b) Hardware Configuration

Configuration	Cluster Node		Standalone Desktop
	Master	Worker	
CPU (Cores)	32	8	12
Memory (GB)	64	32	32
Disk (TB)	24	8	4
Network (Gbit/s)	10	10	10

4.6.3 Privacy and Utility

We discuss the results of running privacy and utility metrics in this section. We illustrated the details of our experiments in Table 4.5.

Table 4.5: Experimental Configurations for Data Privacy and Utility

#	Metrics	Anonymization Parameters	Dataset Size	Platform
1	DM, C_{AVG}, MD, PM	K-*value* $\in$ {2, 5, 10, 25, 50, 75, 100}, $\mid QID \mid = 5$*	Adult = 30K	Spark, Standalone
		K-*value* $\in$ {2, 5, 10, 25, 50, 75, 100}, $\mid QID \mid = 5$*	Irish = 30K	Spark, Standalone
2	KLD, I_E	K-*value* $\in$ {2, 5, 10, 25, 50, 75, 100}, $\mid QID \mid = 5$*	Adult = 30K	Spark, Standalone
		K-*value* $\in$ {2, 5, 10, 25, 50, 75, 100}, $\mid QID \mid = 5$*	Irish = 30K	Spark, Standalone

* Indicates the total number of attributes. Here there are 5 attributes in the experiment.

4.6.3.1 Privacy Results

The results of KLD metric on Adult dataset are shown in Fig. 4.6(a). The results show that the KLD values stay identical between Spark and standalone environment which means the implementation of data anonymization in Spark didn't affect any privacy level. The KLD values only increased from around K group size 2 to 5. After K-value (i.e., group size) = 5 the KLD values remain the same for the rest of the K group size. The visible increase of KLD from K-value 2 to 5 (and slight changes from 5 onward) is due to the active generalization level being applied. At approximately K-value 10, all generalization has been applied and there are no more changes to the rest of the K-value thus KLD value remains identical.

The results of KLD metric on Irish dataset are shown in Fig. 4.6(b). In general, the overall observation of the changes of KLD values is similar to that of Adult dataset. However, we observe that the average KLD values are much higher in the Irish dataset than Adult dataset. This is due that the Irish dataset has more generalization levels for each QID which increase the chances of more number of QIDs to share the same value. This increases a privacy level.

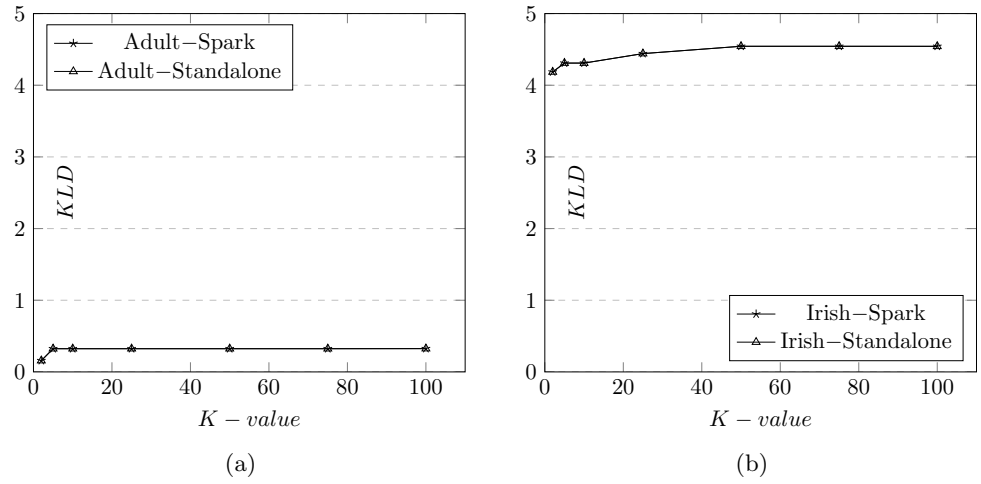


Figure 4.6: Divergence for Adult and Irish datasets on both Spark and Standalone. (**a**) KL -Divergence in Adult Dataset; (**b**) KL-Divergence in Irish Dataset.

The results of I_E metric on Adult dataset are shown in Fig. 4.7(a). Again, the values between the Spark and Standalone remain the same which ensures that the implementation of our data anonymization technique in Spark didn't destroy the privacy level. The average of I_E values in Adult dataset is lower compare to Irish dataset.

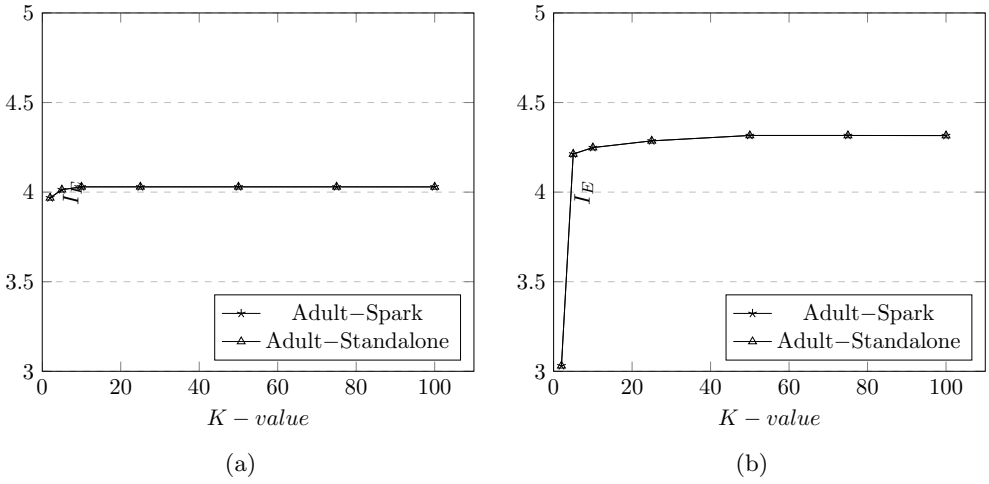


Figure 4.7: Information Entropy for Adult and Irish datasets on both Spark and Standalone. (**a**) I_E in Adult Dataset; **(b)** I_E in Irish Dataset

Our investigation reveals that Adult dataset contains relatively the small number of different QIDs which share the same value as the result of anonymization. The smaller K value affects the I_E value more compare to the greater K value due to the number of same values in QID attributes. This affects in the higher I_E value as it is easier to

identify a unique record within the same equivalent class compare to Irish dataset which has a larger number of different QIDs that share the same value.

4.6.3.2 Utility Results

We illustrate the results of data utility metrics, based on the results obtained from Adult dataset Fig. 4.8(a,c,e,g) and from Irish dataset Fig. 4.8(b,d,f,h).

We first discuss the data utility results of Adult dataset. The overall DM scores produced by both Spark and standalone are relatively high at 0.9. Recall that DM measures the data utility of tuples within an equivalent class. It is expected that the increased in the K group size would result in the increase in the equivalent class. As the equivalent class becomes larger, there will be more changes to make tuples to be more indistinguishable which would result in a high DM score—the results represented in Fig. 4.8(a). In addition, there is a sudden increase in the DM score approximately around $K = 5$ both in the Spark and standalone. This illustrates that at $K = 5$ and onwards the degradation of data has reached the maximum and there is no more generalization/suppression to be applied (i.e., data utility is at the lowest).

The trend of C_{AVG} scores were similar to DM as both metrics were based on the calculation according to the size of equivalence classes. We observe the trend where the data utility scores decline when the size of K group increases as there are more matched distinct attributes. The average penalty seem to remain same at around $K = 10$ with no changes in generalization. The rationale is that at this point, there are no more generalizations or suppressions to apply to an equivalence class. As a consequence, the average penalty for an equivalent class drops when the number of K group size grows. This is seen in Fig. 4.8(b).

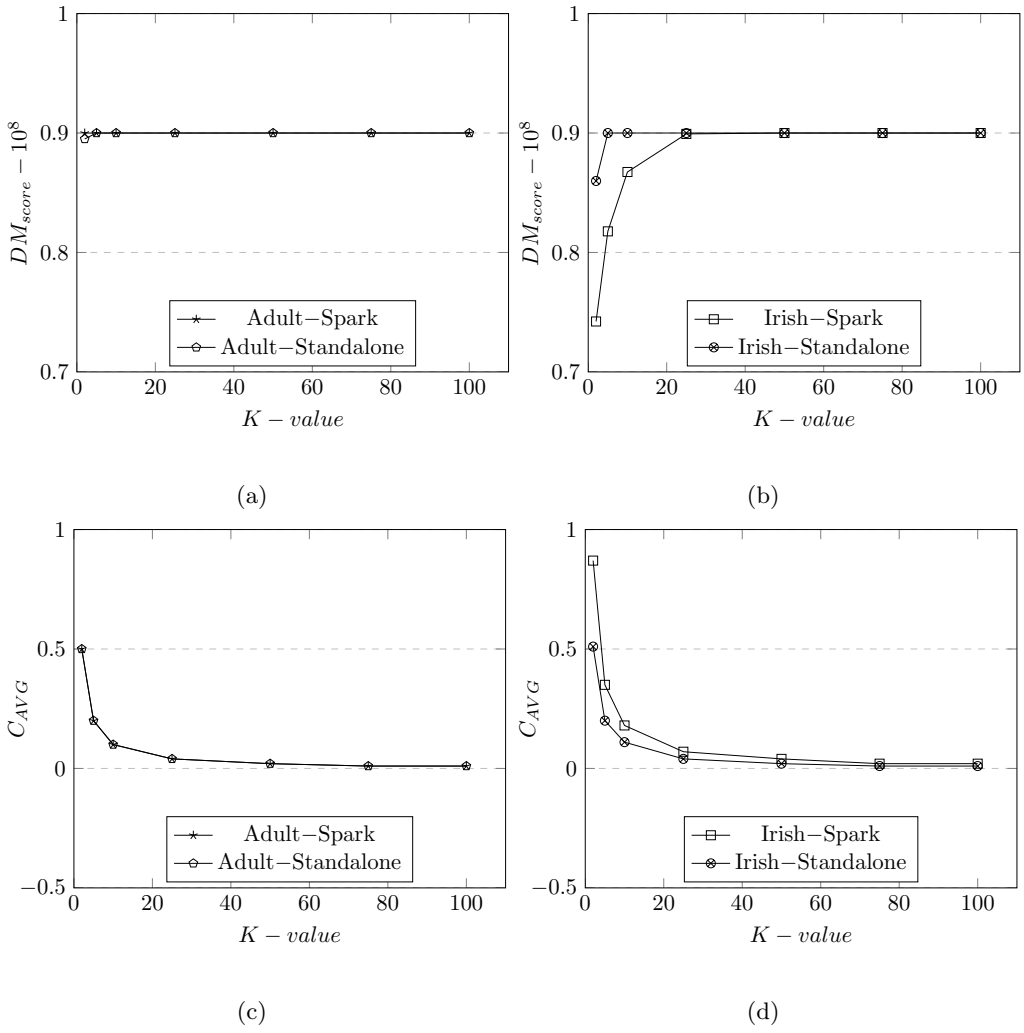


Figure 4.8: *Cont.*

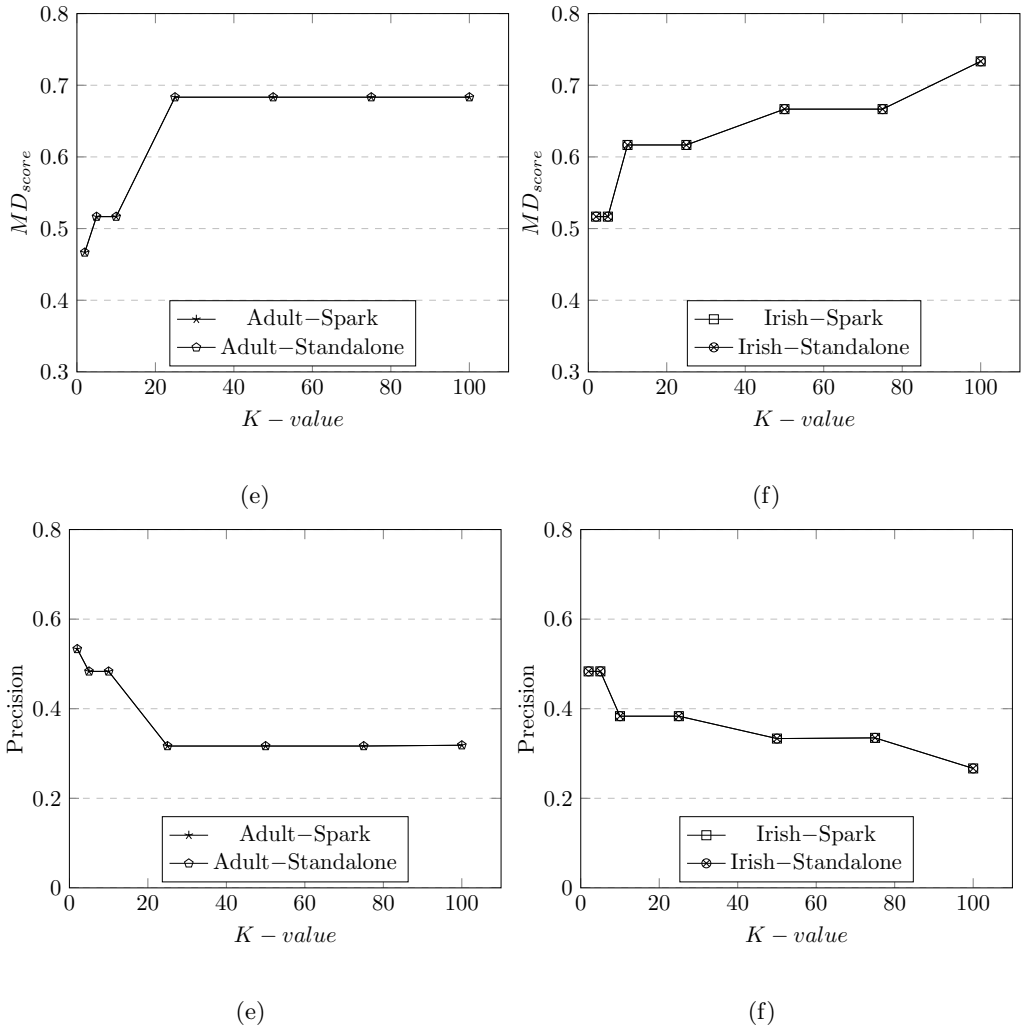


Figure 4.8: Data Utility Metrics for Adult and Irish datasets on both Spark and Standalone. (**a**) Discernibility Metric in Adult Dataset; (**b**) Discernibility Metric in Irish Dataset; (**c**) Average Equivalence Class in Adult Dataset; (**d**) Average Equivalence Class in Irish Dataset; (**e**) Minimal Distortion in Adult Dataset; (**f**) Minimal Distortion in Irish Dataset; (**g**) Precision Metric in Adult Dataset; (**h**) Precision Metric in Irish Dataset.

Fig. 4.8(c) illustrates the results of MD which measures the rate of data utility based on the changes made to tuples from the original dataset to the anonymized dataset. It is expected that MD score would increase when the K group size increases because there would be more attributes in a tuple not matching between the original dataset and the anonymized dataset. MD tends to be more sensitive to generalization levels because the attributes in a tuple applied with higher generalization levels would have more dramatic changes.

Precision Metric (PM), in Fig. 4.8(d), demonstrates the level of distortion at the record level (i.e., the combination of tuples and attributes). It is expected that the PM

score will be higher as the number of K group size increases as there are more records that have lost its original values. The PM score is highly sensitive to GL for each *qid*. This is shown in Fig. 4.8(d) where the PM score increases as the number of K group size increases for both Spark and standalone. This is because the level of GL applied in each *qid* is increased to its highest as the size of K group increases. We observe that at $K = 25$ and onward, the *qid* are appeared to have been generalized to its highest level as the PM score stays the same.

4.6.4 Scalability, Performance and Caching

We ran three sets of experiments to understand scalability, performance, and cache management as shown in Table 4.6. The execution time for running both FlatMapRDD and ReduceByKeyRDD was measured.

Table 4.6: Experimental Configurations for Scalability, Performance and Caching

(K-*value* $\in$ { 10, 20, 25, 50, 75, 100} on Spark)

#	Experiment	Anonymization Parameters	Dataset Size
1	QID Size	A set of $\mid QID \mid \in$ [1, 2, 3, 4, 5, 6]*	Adult = 10M
			Irish = 10M
2	Records Size	$\mid QID \mid$ = 5*	Adult =(5M,10M, 20M, 30M, 40M, 50M)
			Irish =(5M,10M, 20M, 30M, 40M, 50M)
3	Cache Storage Levels	$\mid QID \mid$ = 5*, Memory, Disk, Memory_AND_Disk, OFF_HEAP	Adult = 10M
			Irish = 10M

* It indicates the number of attributes that were used in the experiments.

4.6.4.1 Scalability

In the first set of experiments, we measure the scalability of SparkDA on Adult dataset and Irish dataset by varying the size of QIDs. Before running a scalability test, we first run an experiment for increasing the size of K group on a fixed number of QID to understand the relationship between the execution time and the size of K group. Results show that the execution time appears not to be affected by increasing K group size. This can be explained by following. The number of iterations from the original data to fully anonymized dataset is decided based on the frequency of distinct tuples. The number of K group size would increase the number of tuples. With the fixed number of QIDs, the number of tuples that are increased doesn't necessarily are distinct. This means the frequency count stays the same. With the frequency count remaining the same, the same number of operations are done irrespective to the increasing number of K-size thus the execution time stays the same.

In contrast, as soon as we increase the size of QIDs, the execution time starts to increase. This is because the processing of QID involves applying generalization levels after counting for the number of distinct attribute values which require many iterative operations. Adding more QIDs involved generating more operations. Therefore, the execution time is increasing in the order of the increasing number of QIDs. This is shown Fig. 4.9(a,b).

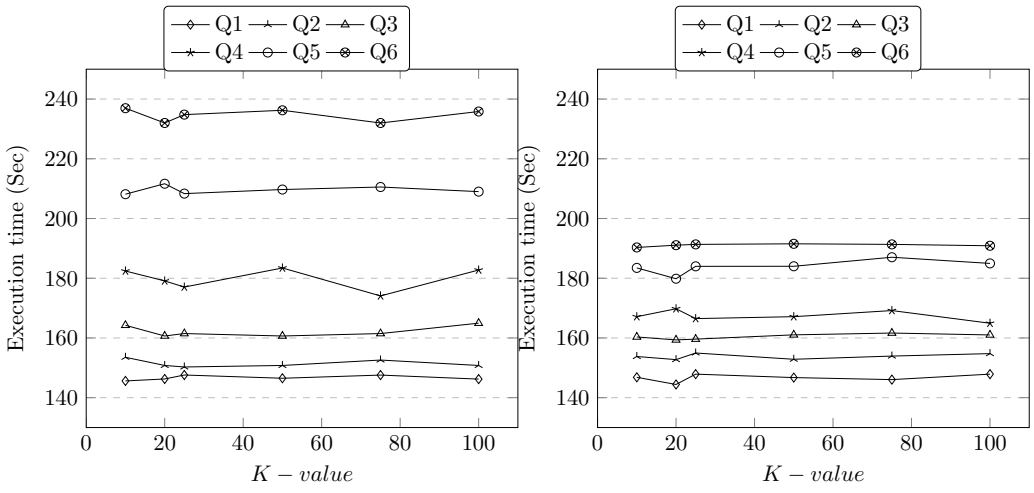


Figure 4.9: Execution Time vs. QID Size. (**a**) Adult Dataset; (**b**) Irish Dataset.

We examine the details of different QIDs from both datasets. It appears that there is a strong performance relationship between the distinctness of quasi-identifiers (i.e., often regarded as cardinality) and the execution time. For example, the execution time has sharply increased between Q4 and Q5 in Adult dataset. We observe that the new attribute "Occupation" in Q5 has a high cardinality and it affected the execution time. In addition, we see that higher execution times in Adult dataset as this dataset appears to have more variations of distinct values.

4.6.4.2 Performance

The second set of experiments is conducted to understand the performance of our proposal. We first compare the performance of our approach against existing data anonymization approaches. The list of existing approaches that were compared include: Spark based multi-dimensional sensitivity-based anonymization (Spark MDSBA) [10], MapReduce based multi-dimensional sensitivity-based anonymization (MR MDSBA) [9], Apache Spark based top-down specialization (Spark TDS) [119], and MapReduce based multi-dimensional top-down specialization (MR MDTDS) [9]. In order to ensure the comparability of results across different approaches, we used the same workload and enforced our configuration to match with the experimental configuration discussed in References [9,10,119] as much as possible.

Fig. 4.10 illustrates the execution time obtained across different methods. As clearly seen, our proposal outperforms other similar approaches by providing the lowest execution time. SparkTDS appears to show the highest execution time. Our analysis demonstrates that SparkTDS updates the score of all leaf which appears to be expensive additional

overhead. This is because the increase in the number of leaves and associated operations (e.g., applying generalization level at leave) naturally demand more execution time especially for higher K-group sizes. The MapReduce-based approaches, seen in MR MDTDS and MDSBA, appear to have a higher execution time mainly due to expensive disk I/O associated with intermediate data. Spark MDSBA performs relatively well when compared to other approaches. We observed that Spark MDSBA uses a larger memory size compare to the dataset size which results in reduced execution time.

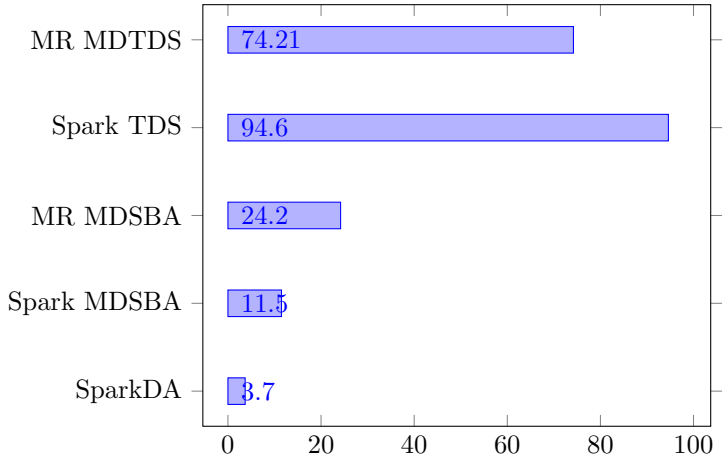


Figure 4.10: Performance comparison with existing approaches.

Secondly, we conducted a performance experiment to understand the impact of execution time against the growing number of records on the fixed size of 5 QID attributes. As seen in Fig. 4.11(a,b), the execution time remains same irrespective to the size of K group. This appears that some operations (e.g., involved in QID generalization) are cached in memory then re-used and this does not affect too much on the execution time. However, this changes as soon as the number of records is increased. The execution time linearly increases as the number of records increase in both datasets.

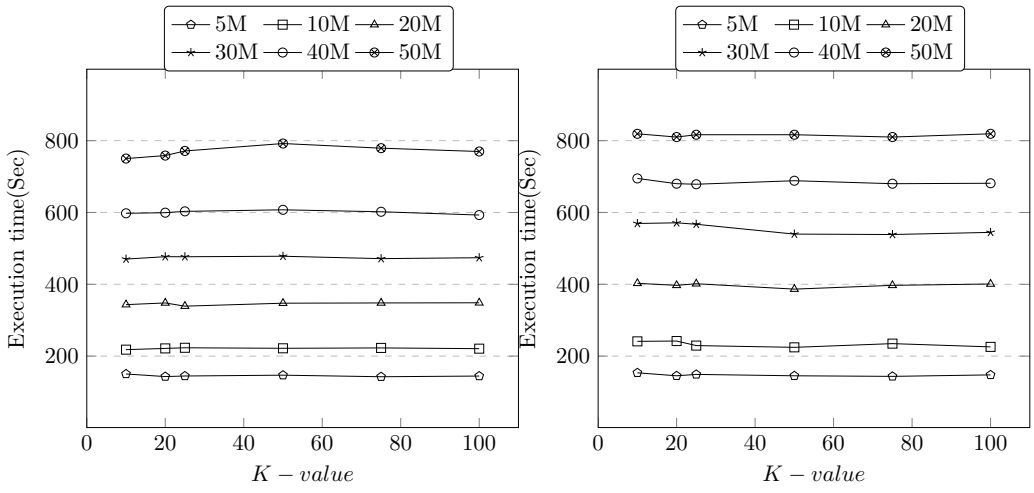


Figure 4.11: Execution Time vs. Record Size. (**a**) Adult Dataset; (**b**) Irish Dataset.

4.6.4.3 Caching

Spark offers a multiple cache storage levels to speed up the process of the same RDDs that are accessed multiple times. The Spark cache strategies can be categorized as follows [116].

- MEMORY_ONLY: RDD partitions are cached in memory only.
- OFF_HEAP: RDD partitions are cached outside the process heap (of JVM) in native memory therefore they are not processed by the garbage collector.
- MEMORY_AND_DISK: RDD partitions are cached in memory. If enough memory is not available, some RDD blocks (usually based on Least Recently Used, or other strategies [144] from memory are written off to disk.
- DISK_ONLY: RDD partitions are cached on disk only.

During the anonymization process, the two RDD transformations we utilize, FlatMapRDD and ReduceByKeyRDDs, are accessed multiple times for generalization from the main application SparkDA. We have set up our experiment with the different cache management options. The results are shown in Fig. 4.12(a,b). In general, the memory-based strategies where the RDD blocks are stored in the memory, such as MEMORY_ONLY and OFF_HEAP, outperformed compared to the cached in disk. Understandably, in-memory inside the JVM cache strategy MEMORY_ONLY took the least execution time compared to out of JVM memory cache strategy used by OFF_HEAP. The MEMORY_AND_DISK took more time than memory-based strategies but less than DISK_ONLY as expected as this strategy allows the switch from memory to disk when

the allocated memory is fully consumed by RDD blocks. Comparing the overall cache performance, the average execution time for Irish dataset was less than Adult dataset. The higher generalization levels for different attributes in Adult dataset has contributed toward the increase in the execution time as there were more ReduceByKeyRDD operations for the generalization levels defined in the *DGH* thus the updates for attributes were more frequent.

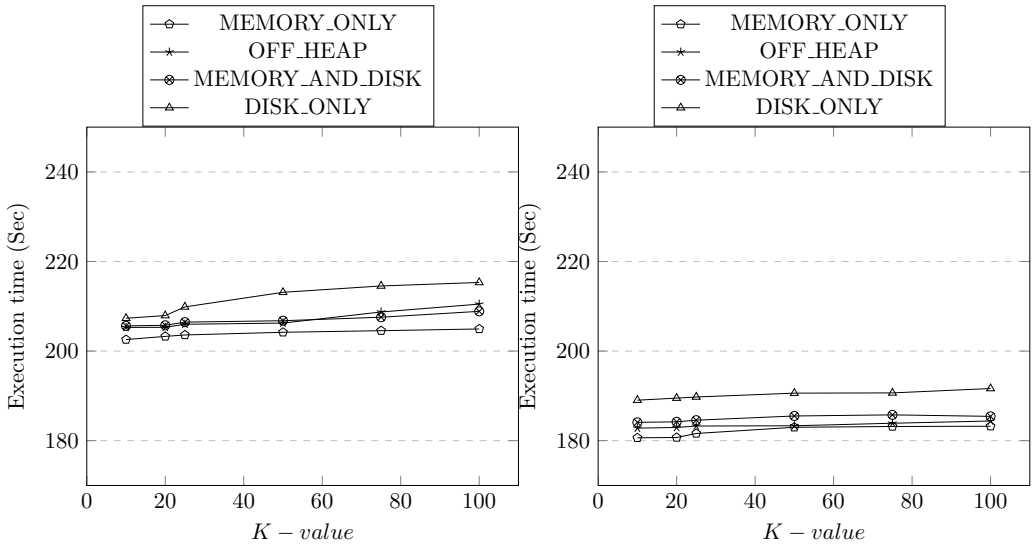


Figure 4.12: Execution Time vs. Cache Strategies. (**a**) Adult Dataset; (**b**) Irish Dataset.

4.7 Conclusions and Future Work

This work introduces "SparkDA" a novel data anonymization approach designed to take the full advantage of Spark platform to generate privacy-preserving anonymized dataset in the most efficient way possible. Our approach is based on two RDD transformations FlatMapRDD and ReduceByKeyRDD with a better partition control, in-memory processing, and efficient cache management. These new innovations contribute towards reducing many performance overheads associated in other similar approaches implemented in MapReduce. The set of experimental results showed that our proposal provides high performance and scalability while supporting high data privacy and utility required by any data anonymization techniques. We also provided insights of a set of performances associated with different memory management strategies offered by Spark and discovered that a side-effect could occur when there are too excessive demands to save data to executor's memory.

In future, we plan to extend our study to implement data anonymization strategy based on the subtree generalization scheme [18]. This new approach will solve the current limitation of the full-domain based generalization approach where attribution values are generalized equally without considering their respective parents' node which results in the loss of data utility to some degree. We also plan to extend our study to implement a more comprehensive data anonymization strategy for multi-dimensional datasets.

Chapter 5

A Generic Approach for Subtree-Based Data Anonymization for Apache Spark

Summary

Among many data anonymization strategies, subtree generalization has been hailed as a technique that provides more efficient generalization strategy compare to full-tree generalization counterparts. Many approaches of subtree generalizations (e.g., top-down, bottom-up, and hybrid) have been implemented under MapReduce platform to take advantage of scalability and parallelism. However, there have been many issues in implementing them in MapReduce due to the lack of support requires for iteration intensive algorithms such as subtree generalization. In this research work, we offer a generic approach for implementing subtree-based data anonymization techniques for Apache Spark. We illustrate how our RDD-based approach resolves the issues associated with MapReduce-based approaches by the use of effective partition management, improved memory usage that utilizes cache for frequently referenced intermediate values, and enhanced iteration support. Our experimental results show that our proposal provides better performance compare to other similar methods while preserving competitive data utility and privacy levels required for any data anonymization techniques.

5.1 Introduction

Subtree generalization provides high data utility and better privacy in data anonymization strategies for single dimensional data when compared to full-tree generalization [48–50]. The iterative nature of subtree generalization is better suited to find more efficient generalization strategy for different attributes. However, such benefit comes with a cost because the complexity of execution time grows as each iteration increases. The complexity would

become more sensitive when other aspects of anonymization are involved, for example a k-group size, the number of attributes, and generalization hierarchy's tree.

Many solutions have been proposed for scalable big data anonymization [24, 68, 152]. Existing approaches of subtree data anonymization are mostly based on MapReduce platforms to take advantage of scalability and cost efficiency [151, 153, 156]. Subtree-based generalization can be broadly categorized into two: Top-Down Specialization (TDS) [156] and Bottom-Up Generalization (BUG) [151]. In TDS approach, the generalization typically starts from the topmost domain values in the taxonomy trees of attributes towards the bottom as an iterative process. In contrast, the techniques based on BUG generalize data from the bottom of the taxonomy tree towards its top, also iteratively. A hybrid approach that combines both TDS and BUG have been proposed [153]. In the MapReduce paradigm, it typically relies on the processing of two primary functions map and reduce where the former works as a sub-unit of data processing while the latter accumulates and produces the final data analytic results. Without an appropriate support for algorithms that runs an extensive iteration such as subtree, the maps and reducers require to communicate many times over, often sequentially and also fetching data from disk, which creates tremendous performance overheads [90, 116, 143].

Spark-based alternative approaches have been proposed to address the overheads associated with MapReduce counterparts, often comparing the performance results on both platforms [90, 116, 143]. In-memory based Spark's performance has well been documented and proven effective for many iteration intensive algorithms such as seen in [143] where it demonstrated 10 times faster performance gain. Other approaches [56, 85, 90, 116] also demonstrate the competitive performance advantage of Spark.

Close to our research, a number of proposals have been emerged to illustrate the use of Spark for data anonymization techniques. For example, [119] proposed a distributed Top-Down Specialization (TDS) algorithm that can work on Spark, and [9, 10] proposed a number of sensitively-based multi-dimension data anonymization strategies. Although these existing proposals offer interesting aspects of the k-anonymity based anonymization strategy, they neither provide any guidelines and strategies as how different type of subtree data anonymization approaches can be best implemented using Spark as a generic framework nor provide any implications of privacy and utility measure.

This research work provides and identify the issues involved in the existing subtree-based generalizations implementation under MapReduce platform and propose a generic subtree-based data anonymization framework implementation using Apache Spark. The main contributions of this chapter are as follows:

- This research work provides a generic RDD-based subtree generalization implementation strategy for Apache Spark and use novelty of our proposal to resolves the existing issues in subtree implementation approaches. We clearly demonstrate how our proposal can reduce the complexity of operations and improve performance by

the use of effective partition; improved memory and cache management for different types of intermediate values; and enhanced iteration support required for generalization process;

- We demonstrate the high-performance outcome of our approach in comparison with existing Spark and MapReduce subtree-based approaches. Our approach outperforms other similar studies in literature with more than 50% less execution time for the same dataset size;
- This research study provides various experimental results to illustrate our approach processing compatibility on real life and synthetic datasets. We also compute the performance, scalability, and anonymity validations on various constrains, such as: Spark cluster configuration(computing and storage), Dataset size (number of record and attribute numbers), and anonymity benchmarks(privacy and utility) to illustrate the application of our proposed model.

The rest of this chapter is organized as follows. Section 5.2 provides a related work to ours and highlights the pros and cons of each similar work. Section 5.3 provides the background and definition used throughout the chapter. Section 5.4 discusses the details of the issues involved in existing subtree generalizations implemented in MapReduce. Section 5.5 describes the details of our proposal and clearly illustrates how our proposal can resolve the issues associated with the MapReduce-based approaches. In Section 5.6, we provide our experimental results including setup, configuration, and discuss the observations of the results. Finally, we conclude the chapter in Section 5.7 and provide some potential future directions.

5.2 Related Work

The majority of existing approaches have been implemented as sequential MapReduce jobs where the output of each MapReduce job is used as an input for subsequent steps until the anonymization constraints met. Such sequential execution of jobs can attribute significant performance overheads. A number of Spark-based approaches have been proposed to address the concern associated with MapReduce-based data anonymization strategies. Zaharia et al. [143] illustrated a competitive performance advantage of in-memory based Spark operations compared to disk based MapReduce execution. Their results demonstrated that Spark implementation of iterative operations was 100 times faster than it was implemented under MapReduce platform as Spark provides better parallelism by allowing many iterative tasks running at the same time often accessing memory instead of disks. The authors in [116] provided benchmarking results using Word Count, k-means and PageRank where Spark outperformed over MapReduce especially on iterative tasks. Their work stated that the performance gain of Spark was due to RDD caching that

reduced the overheads associated with disk and CPU. Maillo et al. [90] demonstrated the performance advantage of Apache Spark on iterative tasks based on K-nearest Neighbour (KNN) using the datasets which contained 10 million instances.

An anonymization approach for Spark has been design by Sopaoglu and Abul [119], The authors developed a distributed Top-Down Specialization (TDS) algorithm to provide k-anonymity in Apache Spark. The main focus of their study was to improve the scalability aspect of the original TDS algorithm [156] by offering an improved partition management. Using the adult dataset, the authors evaluated that the scalability and run-time were significantly improved, however, the authors did not provide the privacy and utility outcomes of larger datasets. Al-Zobbi et al. [9, 10] proposed a number of sensitivity-based multi-dimension anonymization strategies that could produce different levels of the information obscurity depends on the different access privilege levels of the users (i.e., more customized data generalization result suitable for each user). To understand the roles and responsibility of the user accessing the system, the proposal utilised a User Defined Function (UDF) of Spark which allows the developer of Spark to be able to extend the vocabulary of default Spark SQL. Their proposal also illustrated that it was possible to reduce the data transmission time between memory and disks by serializing data with Spark RDD. However, the proposed solution performance degrades for larger dataset size compare to computing memory allocated.

To address the overheads associated with MapReduce, a number of Spark based approaches have been proposed in recent years [13, 20, 69, 103]. Reference [28] proposed the INCOGNITO framework for full-domain generalization using Spark RDDs. Though their experiential results illustrate the improvement in scalability and execution efficiency, their proposal does not provide any insights of privacy and utility trade-offs. Anonylitics [107] utilized Spark's default iteration support to implement data anonymization. However, their approach does not address the potential memory exhaustion unable to accommodate increasing number of intermediate data produced as the number of iteration increases. PRIMA [11] proposes an anonymization strategy for Mondrian algorithm with Optimal Lattice Anonymization (OLA) which is used to define the utility and generalization level rules in order to limit the data utility loss. However, the study uses the Mondrian algorithm in standalone implementation rather then parallel processing. Reference [97] proposes a distributed Mondrian approach by splitting the input data to the partitions allocated to each node of cluster by using Spark k-mean. A series of Spark jobs runs on each cluster node to produce anonymized results, while the anonymized results are merged together later by another cluster node. However, the study does not provide benchmarks to compare to results for scalable size to validate the impact of distribute data on their proposal. The study [12] provides proposed DI-Mondrian in Apache Spark platform to provide L-diversity privacy. The proposal compare the anonymization of adult dataset with MapReduce based approach, however, the study does not validate the scalability and performance impact of large dataset for spark platform.

5.3 Subtree Generalization

In this section, we describe the basic symbols and their descriptions used in this chapter (see Table 5.1) together with the general algorithm involved in a subtree generalization.

5.3.1 Preliminaries

Let define a dataset $D = \{r_0, r_1, ..., r_{n-1}\}$ as a set of data records r_i where $0 \leq i < n$ and $|D| = n$ denotes the total number of records in a dataset. Then, a record $r \in D$ can be constructed by a set of attributes $A = \{a_1, a_2, ..., a_m\}$ and each record consists of multiple attribute values $r = (av_1, av_2, ..., av_m)$ where a_j and av_j denote the j^{th} attribute and the attribute values of a record respectively, where $0 < j \leq m$, and m denotes the number of attributes in the dataset D.

Table 5.1: Symbols and Descriptions

Symbol	**Description**	**Reference**
D	Dataset	Algorithm 10
D^*	Anonymized Dataset	Algorithm 10
r	Record	Algorithm 10– 13
SA	Sensitive Attributes	Algorithm 10
RDD_in	Input_RDD	Algorithm 11
RDD^*	Anonymized_RDD	Algorithm 13
$\|D\|$	Total number of Record	Algorithm 13
n	nth Record	Algorithm 10– 13
r^*	Anonymized Record	Algorithm 13
AL	Anonymization Level	Algorithm 13,12
C_r	Record count	Algorithm 11,12
C_r^*	Anonymized record count	Algorithm 13
A_v	Attribute Value	Subsection 5.5.2
TT	Taxonomy Tree	Algorithm 12
K	Anonymity Parameter	Algorithm 13
QID	Quasi-identifiers set	Algorithm 13
qid	Quasi-identifier	Subsection 5.5.2
C_A	Child attribute	Algorithm 12
P_A	Parent attribute	Algorithm 12
DOM	Domain value in TT	Algorithm 12

Definition 1 (Quasi-Identifiers)**.** A Quasi-Identifier $QID \subseteq A$ in a dataset can be defined as a set of attributes for a record which is available to public such that the value of each single attribute $a_j \in QID$ cannot identify an individual, but a combination of attributes in QID may lead to unique identification for an individual such that:

$$(\bigcup_{j=1}^{v} a_j) \in QID,$$

where $v \leq m$ while a_j is called as qid. For instance, an instance {9th, M, 32} is a QID while each attribute within a QID such as {9th}, { M}, or {32} is a qid.

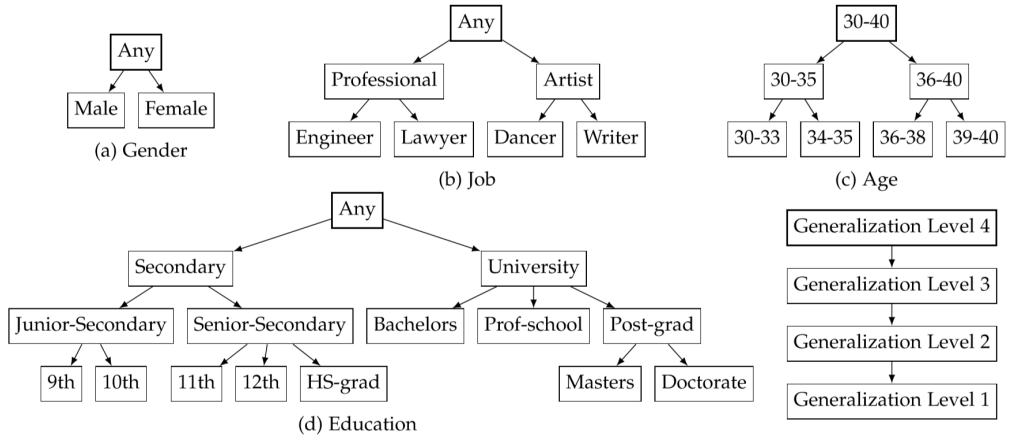


Figure 5.1: Some examples of generalization taxonomy trees for gender, job, age, and education

Definition 2 (Generalization)**.** Generalization refers to replacing the original value of the attributes of $QIDs$ with the generalized (less specific) values. For example, the original value of the age [32] is replaced by the generalized value [30-35].

Definition 3 (K-anonymity)**.** The dataset D is K-anonymous if $k > 1$ and for every single record $r_i \in D$ there exists at least K-1 other records that share the same QID. In other words, the size of each Equivalent Class in the dataset should be at least K, where the Equivalent Class is a set of tuples with the same QID which each $a_j \in QID$ has the same attribute values (a_v).

Definition 4 (Sensitive Attributes)**.** Sensitive Attributes ($SA \subset A$) can be defined as a set of attributes that should remain private so that an adversary cannot deduce their values.

Definition 5 (Taxonomy Trees)**.** Taxonomy Trees (TT) are the logical hierarchies of distinct values in a dataset. Taxonomy Trees are provided by either the data provider or the data recipient for all QID in the dataset. The examples of taxonomy trees are shown in Fig. 5.1 .

Definition 6 (Cut)**.** The attribute values of a generalized QID form a "cut" using Taxonomy Tree TT. A cut of a tree is a subset of values in the TT that contains exactly one value on each root-to-leaf (parent to child) path. The cut can be used to decide the depth of anonymization one wants to achieve [50].

Definition 7 (Anonymization Level)**.** Anonymized Level (AL) represents the total number of generalized records in a (semi-)anonymized dataset with the respect of K-group size records. AL can be used to decide whether the original dataset requires further anonymization process applied.

5.3.2 Subtree Generalization Algorithm

In this section, we describe a generic subtree generalization algorithm using a sample dataset.

As mentioned, Fig. 5.1 represents the example of Taxonomy Trees (TT) based on Gender, Age, Job, and Education of the census dataset [15]. Each TT includes roots (parent nodes), middle nodes (in between the parent and child nodes but most often act the same as the parent nodes), and leaves (which are mostly child nodes). In a subtree scheme, generalizations are applied for the parent nodes if any child nodes are generalized. For example, in Fig. 5.1(b), if 'Dancer' child node is generalized to its parent node 'Artist', then other child node 'Writer' also needs to be generalized to 'Artist'. Note that 'Engineer' and 'Lawyer' child nodes retain their values as the dimension of their parent node "Professional" is not affected. The root (parent) node of all taxonomy trees are often called 'Any'.

Subtree generalizes data by applying one level of generalization at a time on an attribute by converting child node to parent node. The Subtree generalization steps are presented in Algorithm 10. The iteration starts from the child level. Then, at each step, a specific value (i.e. child) is generalized to a general value (i.e. parent) for an attribute within a QID. This process is repeated until the highest level of generalization violates K-anonymity rule [127] (explained in Definition 3). Table 5.2 shows the original dataset

Table 5.2: A sample dataset

Education	**Gender**	**Age**	**Income**	**Count**
9th	M	30	$\leq$ 50k	3
10th	M	32	$\leq$ 50k	4
11th	M	35	$>$ 50k	2
11th	M	35	$\leq$ 50k	3
12th	F	37	$>$ 50k	3
12th	F	37	$\leq$ 50k	1
Bachelors	F	42	$>$ 50k	4
Bachelors	F	42	$\leq$ 50k	2
Bachelors	F	44	$>$ 50k	4
Masters	M	44	$>$ 50k	4
Masters	F	44	$>$ 50k	3
Doctorate	F	44	$>$ 50k	1

Table 5.3: The result of the first generalization applied to the original dataset

Education	**Gender**	**Age**	**Income**	**Count**
Junior-Secondary	M	30	$\leq$ 50k	3
Junior-Secondary	M	32	$\leq$ 50k	4
11th	M	35	$>$ 50k	2
11th	M	35	$\leq$ 50k	3
12th	F	37	$>$ 50k	3
12th	F	37	$\leq$ 50k	1
Bachelors	F	42	$>$ 50k	4
Bachelors	F	42	$\leq$ 50k	2
Bachelors	F	44	$>$ 50k	4
Post-grad	M	44	$>$ 50k	4
Post-grad	F	44	$>$ 50k	3
Post-grad	F	44	$>$ 50k	1

along with the count of each record (i.e., the frequency of the same record appeared) in the database. Table 5.3 is produced from Table 5.2 as a result of a generalization level applied based on Taxonomy Trees depicted in Fig. 5.1. After the first level of generalization, we observe that the attribute of Education for the child nodes "9th" and "10th" are generalized to "Junior-Secondary". Similarly, "Masters" and "Doctorate" child nodes are generalized to "Post-grad", and other child nodes remain the same in this round.

Table 5.4: Fully anonymized dataset applied to the original dataset

Education	Gender	Age	Income	Count
Junior-Secondary	M	30 – 33	$\leq$ 50k	7
11th	M	35	$>$ 50k	5
12th	F	37	$>$ 50k	4
Bachelors	F	40 – 45	$>$ 50k	10
Post-grad	Any	44	$>$ 50k	8

Finally, this iteration process is repeated until all QID meet the final required anonymization level, as represented in Table 5.4.

Algorithm 10: Subtree Generalization Algorithm

Input: D
Output: D^*

1 QID, $SA \leftarrow r$ for i=1 to n in D;
2 Compare k with Count of r ;
3 Compare Cut_i score for each QID_i select highest;
4 Replace QID_i child to QID_i parent in TT for Cut_i score;
5 Count the r with updated value;
6 Repeat step 2 to 5 until k is greater anonymized Record ;
7 **return** *(D^*);*

The overall subtree generalization algorithm is described in Algorithm 10. Each round of iteration includes four major steps: (i) Comparing the K-anonymity level with the number of records generalized, (ii) Calculating the data utility and privacy scores based on [50] for all $QIDs$, (iii) Finding the best generalization level by comparing the score values for all QIDs and decide the next generalization level based on the highest score of a QID, and (iv) Applying the highest score of the QID and apply the generalization to all QIDs in the same Equivalence Class.

5.4 Review of subtree Implementation in MapReduce

In this section, we review the subtree implementation based on MapReduce platform and extensively discuss the main limitations involved. There are four main phases in a typical subtree implementation that use MapReduce platform [151] (shown in Fig. 5.2). MapReduce jobs contained in these four phases are coordinated together to accomplish the subtree anonymization. The description for each of the four main phases is as follows:

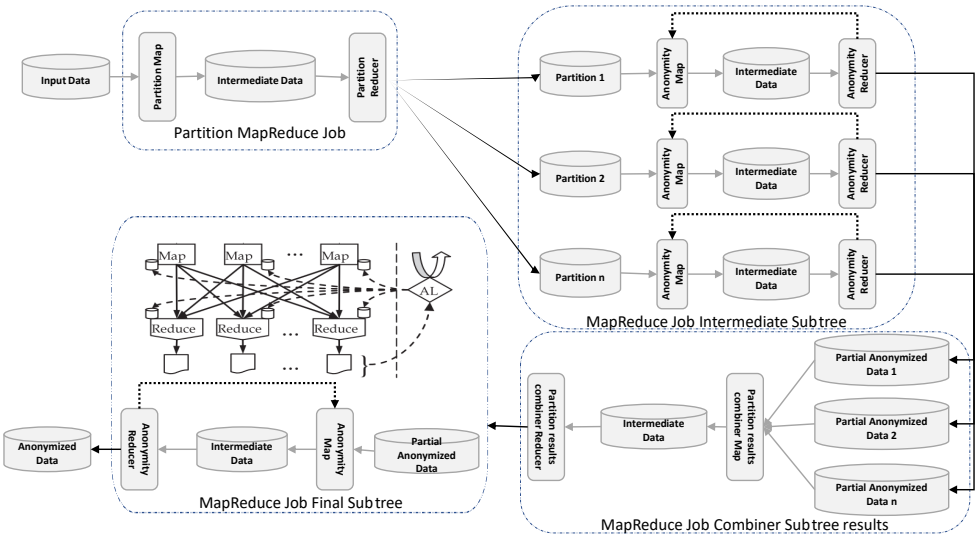


Figure 5.2: Subtree data-flow diagram in the MapReduce platform

(1) *Partition MapReduce Job*: this phase involves dividing the original datasets into multiple chunks (i.e., partitions) in which each chunk contains a smaller portion of the original datasets.

(2) *MapReduce Job Intermediate Subtree*: This phase applies data anonymization to each chunk in parallel resulting in producing intermediate anonymized results.

(3) *MapReduce Job Combiner Subtree Result*: In this phase, MapReduce jobs combine all intermediate anonymized results to form an intermediate anonymized dataset.

(4) *MapReduce Job Final Subtree*: In this phase, the K-anonymity for the complete dataset is validated using the execution of two MapReduce jobs on the intermediate anonymized datasets.

The bold solid arrow lines represented in Fig. 5.2 indicate how *parallelism* works in MapReduce platform. It shows the timeline of the parallel processing for multiple MapReduce jobs which are executed in parallel on each node to process the intermediate data. The grey solid arrow lines show the MapReduce platform *execution flow* from mapping intermediate data to reducing phase while the dashed arrow lines describe the *data flow* from dispatching Anonymization Level (see Definition 7) into a cache for updating K at each iteration MapReduce round.

However, we identify the following architectural limitations of MapReduce platform for implementing subtree anonymization algorithm. These include the issues associated with partition, memory, and iteration management. We argue that these limitations create

execution complexity and performance degradation in various stages. We discuss these problems in detail in the following sections.

5.4.1 Partition

Processing data in MapReduce requires Map task to process a portion of the input data by assigning key-value pairs followed by generating an intermediate data. The intermediate data is stored in a local disk of each executor node after applying hash function. The hash function generates the key-based order by assigning and storing values to its respective keys. The hash applied to each partition ensures that the output of a Map task is arranged using sort and shuffle process. This hash order ensures Reducers can access its respective key-value pairs based on intermediate data locality [40].

An uneven hash partitioning of intermediate values may create skew data in multiple places. For instance, tuple skewness [156] may happen in a situation where a node contains a proportionally larger number of records than other nodes. As a consequence, a reducer coordinating these multiple nodes to process the outcomes now will have to wait for a significant time until the node containing the larger number of records completes. Similarly, key skewness [60] may happen when there is a big difference in the generalization levels being applied to the values of records (e.g., applying a single generalization level verses multiple generalization levels). The overhead of having to deal with such key skewness would scale up rapidly when K-group size increases.

To put more formally, let n be the number of tuples and m be the number of attributes in a dataset D, and let s and t represent the number of Mapper and Reducers, respectively. Then, Mapper produces $m + 1$ key-value pairs which yields $\mathcal{O}(1)$ space and $\mathcal{O}(m * n/s)$ time complexity [129]. However, the Reducer yields $\mathcal{O}(1)$ and $\mathcal{O}(m/k * n/t)$ for space and time complexities respectively, where K denotes a K group size. The increase in s causes less number of n, which reduces the computing time for Mapper process, and by increasing the number of mappers (s) we will get better big $\mathcal{O}$ complexity because $m * n$ is divided by the number of mappers (s) [149, 151]

5.4.2 Memory

The MapReduce I/O operation is done via accessing a stable storage such as disk. As a Mapper loads the input data from disk to memory (of the execution node), the results (i.e., intermediate data) are transferred and stored from the memory to the disk (of the same node). The Reducer loads the intermediate data into the memory again (from the disk) of the execution node where the Reducer runs on to process and subsequently store the results back to the disk [40]. With no support for cache, any values that are produced in different stages (i.e., input, intermediate or output) need to be stored in the disk when read to the memory each time the data needs to be processed. This architectural design

of MapReduce adds excess overhead for I/O operations and needing for larger storage capacity [26, 31, 72].

We will put this more formally. Let subtree uses N_{It} for non-iterative jobs, and It for iterative jobs to convert dataset to anonymized data. J represents a MapReduce job, every J reads R times from the disk and W times writes on disk. I represents the number of iterations needed for each J. Then, I depends on multiple factors including the number of attributes, K group size, and generalization hierarchy. Following equation is used to calculate the total number of R and W operations in MapReduce subtree (ST).

$$ST(R,W) = \underbrace{\left[\sum_{J=1}^{N} J \cdot (W+R)\right]}_{N_{It}} + \underbrace{\left[\sum_{J=1}^{M} J \cdot I \cdot (W+R)\right]}_{It} \tag{5.1}$$

Anonymization process causes both more execution time and complexity especially in the Reducer phase while processing intermediate anonymized datasets. The worst case of complexity in the Reducer phase can be calculated as:

$$\mathcal{O}\left(\left(\frac{m}{k*GL}\right)^2 .log\left(\frac{m}{k*GL}\right)\right)$$

In the meantime, the space complexity of the Reducer phase can be formulated as:

$$\mathcal{O}\left(\frac{n}{k*GL}\right)^2,$$

where GL denotes generalization level in K-group.

5.4.3 Iteration

We argue that there are two architectural design principles of MapReduce platform that creates significant overheads for any iteration tasks. The first one is related to the I/O principle where any intermediate results has to be written to disk and subsequently read by the executor memory as we discussed in Section 5.4.2. The second one is related to the data locality principle. The locality principle of MapReduce platform dictates that any computation process on data must be performed based on the data location in the cluster node. As a consequence, the result of the computation process also has to be saved in the same cluster node. The problem arises when the data read is required by other cluster nodes. In this case, the message exchange is required over the network which could cause noticeable delay and will be multiplied by each iteration process.

With the disk I/O-based operation and data locality principles, we argue that any algorithm that involves intensive iterations such as the subtree generalization can cause significant overheads at multiple places such as at *Disk I/O, Network*, and *Scheduling* [72].

Disk I/O overhead: Significant I/O overheads may occur at the stages of subtree generalization where the stage involves an intensive iterations such as applying generalization

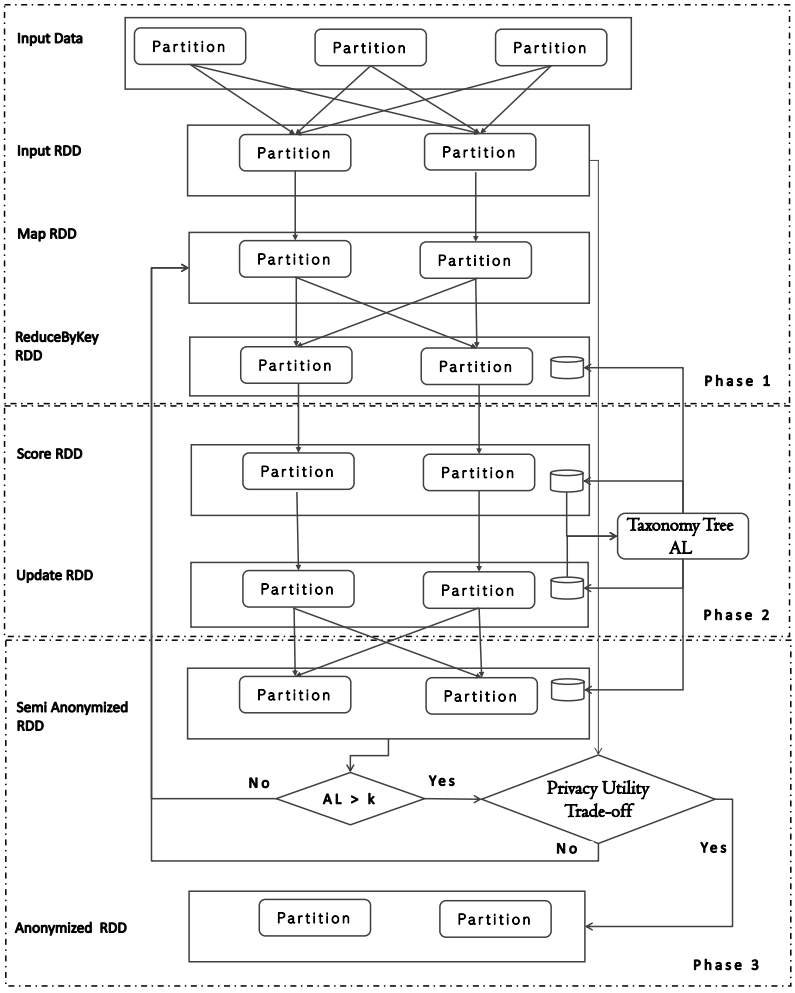


Figure 5.3: Proposed Spark subtree model

levels for attributes, calculating privacy and utility scores, finding the most optimal generalization level, and re-applying the generalization based on the optimal generalization level [84].

Network Overhead: The anonymization steps in MapReduce require to use network to exchange the intermediate data among the cluster nodes. In this case, the network overhead may be created, as the various intermediate data generated by the iterative tasks may need to be transferred to the other cluster nodes multiple times. This problem may get worse by any network delay and consequently is considered as an expensive task causing significant delay in the iteration process [130].

Scheduling Synchronization Overhead: Assume a situation in which there are two mappers with different workloads and one mapper takes a significantly longer time to complete. In this case, the reducer processing the results of these two mappers needs to wait until both mappers complete their jobs. This is referred as a scheduling synchro-

nization. However, if there are many mappers in iterations where the difference in the workloads are observed, the scheduling synchronization overhead can be increased as the number of imbalance across mappers happens [111].

5.5 Our Proposal

This section provides the details of our proposed approach for Spark based subtree anonymization. We discuss the main phases involved in our proposal and describe how we address the concerns discussed in the previous section at each of the phase.

Our proposed approach consists of three main phases:

- *Phase 1 (Initialization)*: This phase ensures that each partition contains the optimal number of records without duplication so that the workload of each partition is balanced. The original data records are counted and then assigned with a frequency value based on the times of appearance of that specific record in the whole dataset. The total record counts is used to address both tuple and key skewness problems discussed in Section 5.4.1.
- *Phase 2 (Generalization)*: This phase calculates the privacy and utility scores for each attribute. The privacy and utility scores are used to find the most optimal generalization level to be applied for a certain attribute. Frequently referenced intermediate values (e.g., the privacy and utility scores and the results of the generalization level being applied) are stored first in memory and then moved to a cache to reduce any potential I/O overhead discussed in Section 5.4.2 and 5.4.3.
- *Phase 3 (Validation)*: This phase validates if generalized dataset meets the K-anonymization constraint. Again, frequently referenced intermediate values (e.g., semi-anonymized dataset) are saved either in memory or cache to reduce the overheads discussed in Section 5.4.2 and 5.4.3

Fig. 5.3 illustrates our proposed approach that utilizes these three main phases. We discuss the details of each phase and specific improvements we have made to resolve MapReduce-based issues.

5.5.1 Phase 1- Initialization

In this phase, we provide a new partition management that can avoid both tuple skewness and key skewness. The following steps detail our partition strategy.

- To avoid the tuple skewness, we first count the total number of records from the input data then divide the records according to the number of partition so that each partition contains roughly the similar number of records.

- To avoid the key skewness, we count the duplicate records that appear in multiple partitions. Their frequency is recorded in one partition and the duplicated records from other partitions are removed.
- After key skewness is addressed by above step, we count the number of records from each partition again (as some duplicate records removed) and move the records across partitions so that each partition contains the similar number of records.

Algorithm 11: Initialization Phase of Spark subtree.

Input: D
Output: $(r, \sum C_r)$

1 $RDD\_in \leftarrow$ RDD (D, Partition factor);
2 Map_RDD (r,C_r) $\leftarrow$ Map (RDD_in);
3 ReduceByKey(r, $\sum C_r$) $\leftarrow$ ReduceByKey (Map_RDD, Partition factor);
4 **return** $(r, \sum C_r)$;

With roughly the equal number of records contained in each RDD partition, each partition executes in parallel by taking approximately the similar processing time. Note that we also consider the capability of each node when we partition (e.g., memory size and CPU capacity) along with the number of records.

We present this initialization step involving efficient partition in Algorithm 11. In Step 1, "partition factor" indicates the variable that contains the number of records and the capability of node. Step 2 uses Map_RDD to transform the input RDD_in as a key, and the value showing the key-value pairs used to process the data such as (r,C_r), where r represents records and C_r denoted the count in each Map. At this phase, the key-value pair is used as the key representing a record while the value holding the number of times in which that record is appeared in the dataset. The ReduceByKey_RDD in Step 3 reads the Map_RDD key-value pairs (r,C_r) and aggregates the value across the same key. Then, the count of the same r is summed up together to find the $\sum C_r$ which represents the total number of record counts across all partitions. Note that this process requires shuffling the data from different partitions in the executor nodes to exchange the values for the same key over the network.

5.5.2 Phase 2- Generalization

The purpose of this phase is to apply the most optimal generalization level according to the privacy and utility scores. We use memory and cache to hold the intermediate results (such as the privacy and utility scores of each attribute, the results of the generalization level applied) to avoid expensive disk access.

We describe the details of this phase in Algorithm 12. Step 1 assigns A_v as child C_A in r for the generalization level for QID while P_A is assigned to all QID based on its C_A in

TT. This process applies one level of generalization for one attribute (i.e., one iteration) and holds the results *in memory* so that the results can be used in the subsequent step.

Steps 3–5 are used to compute the privacy and utility scores which is denoted as $Score_{ILPG}$ (QID) as following, based on [46,47].

$$Score_{ILPG}\,(QID) = \frac{IL\,(QID)}{PG\,(QID)+1}, \tag{5.2}$$

where $IL(QID)$ contains the result of information loss for QID while $PG(QID)$ contains the result of privacy gain for QID. The details of the calculation for $IL(QID)$ and $PG(QID)$ are depicted in Equation 5.3 and 5.4, as follows, respectively.

$$IL\,(QID) = En\,(P_A) - \sum_{C_A \in P_A} \left(\frac{|C_A|}{|P_A|}\right) En\,(C_A)\,, \tag{5.3}$$

where $|C_A|$ represents the child attribute and $|P_A|$ represents the parent attribute for the given QID. $E_n(C_A)$ denoted the entropy value of the child attribute while $E_n(P_A)$ denotes the entropy value of the parent attribute.

$$PG\,(QID) = A_{P_A(QID)} - \ A_{C_A(QID)}, \tag{5.4}$$

where $A_{P_A(QID)}$ contains an Anonymization Level (AL) of the parent QID and $A_{C_A(QID)}$ contains an Anonymization Level (AL) of the child QID.

Note that this privacy and utility score computation requires a multiple iterations depending on the depth of generalization levels to be applied based on the TT. To avoid the disk access, the privacy and utility score for QID and the result of a generalization level applied are stored in *cache*.

Algorithm 12: Generalization Phase of Spark subtree.

Input: (r,$\sum C_r$), TT
Output: (r^*,$\sum C_r$)
1 $C_A \leftarrow A_v$ in r ;
2 $P_A \leftarrow C_A$ (DOM) in TT ;
`/* For all child attribute assign parent attribute */`
3 **for** *C_A and P_A of r in TT* **do**
4 score(QID) ← IL(QID), PG(QID);
5 *RDD_score* ← score(r), C_r ;
6 **if** *score QID is Max(QID)* **then**
`/* Max(QID) decide which child qid need to replace with parents qid */`
7 update r for QID of $P_C \leftarrow P_A$ (DOM) in TT ;
8 Update(score) ← update r ;
9 update AL ← Update(score) ;
10 *RDD_update* (r^*, $\sum C_r$) ← Update(score) ;
11 **return** (r^*, $\sum C_r$);

Steps 7–10 are used to identify and update the best generalization level based on the privacy and utility score calculated in the earlier steps. Note that *RDD_update* in Step 10 requires to access the TT for each r to find a generalization level to apply. In order to expedite this process, we store the TT in the distributed cache of each node.

Note that we also cache AL for the next iteration. The process goes through each r iterating over each A_v, where any A_v belonging to QID is considered as *qid*. The A_v is considered as C_A when the value is compared in TT. The C_A is compared with the same QID attributes in DOM. Once the C_A is found in TT, the C_A value is replaced by its P_A parent nodes. Then, the A_v values are replaced from C_A to P_A for each r to obtain r^*. Finally, the RDD returns anonymized key-value pairs (r^*,$\sum C_r$).

It must be noted that most of data is stored and fetched from memory rather disk during any iteration processes which avoids unnecessary disk I/O overhead. We also utilize the capability of cache in this generalization phase to avoid the re-computation of the intermediate values (i.e. the privacy and utility score and the results of generalization level) during the iterations.

5.5.3 Phase 3- Validation

The purpose of this phase is to validate if the full anonymization has been achieved - that is, the optimal generalization levels for all attributes have been applied up until they do not violate the K-anonymization constraint.

Again, we store frequently used intermediate results such as attributes with a certain level of generalizations applied (also called as semi-anonymized attributes) into cache to avoid expensive disk access and improve memory management.

Algorithm 13: Validation Phase of Spark subtree.

Input: (r^*,$\sum C_r$),k
Output: (RDD^*)
1 ReduceByKey (r^*,$\sum C_r^*$) $\leftarrow$ *RDD_update* ((r,$\sum C_r$), Partition factor);
2 **if** ($AL \leq k$) **then**
3 ($\sum C_r \leftarrow \sum C_r^*$);
4 ($r \leftarrow r^*$);
/* return to Map RDD in phase 1 if need more generalization */
5 **return** (r, $\sum C_r$);
6 **else if** ($AL > k$) **then**
7 ;
8 **for** C_r^* *in* $\sum C_{r^*}$ **do**
9 $r^* \leftarrow (r^*, C_r^*)$;
10 **return** (r^*);
/* completely anonymized */
11 Update $RDD^* \leftarrow r^*$;
12 **return** RDD^*;

The details of this phase is depicted in Algorithm 13. Step 1 is used to update the partition based on the Phase 1 strategy. Steps 2–5 are used to check whether $\sum C_r^*$ that contains the total number of generalized records (represented as AL) meets the K-anonymization constraint or not. If AL has the smaller number of records than K-anonymization constraint, the semi-anonymized records $\sum C_r^*$ needs to be copied to a new partition of a Map $\sum C_r$ and returns to the Phase 1. Steps 6–9 are used in the case where the full generalization is achieved - that is, the number of generalized records meets K-anonymization constraint. Then, a key is assigned for each of (distinct) fully generalized record where the value of fully generalized record is used as a value. Finally, Step 10 saves all fully anonymized records to memory.

Based on the proposed algorithm described in this phase, we mitigate the *disk I/O, network I/O,* and *synchronization overheads* during the iteration involved in this phase. For instance, by saving semi-anonymized dataset in a memory, we reduce the *disk I/O overhead*. Moreover, we minimize any chances for a potential network transfer by reducing the size of dataset through removing duplicate records while still preserving the count and performing RDD operations which share the cached intermediate values without expensive message exchanges across multiple network nodes. This significantly reduces *network I/O overhead*. Because of the optimal number of datasets operated in this level (as the result of partition management descried in Phase 1) reduces *synchronization overhead* significantly as the number of iteration increases.

5.6 Experimental Results

In this section, we first describe our experimental setup including the details of the dataset and the system environment configurations. Then, we provide the results of the memory and iteration performance. Finally, we discuss the results of the privacy and utility scores obtained through a several privacy and utility measurement metrics.

5.6.1 Dataset

We used US Census dataset (often described as Adult dataset) [15]. We downloaded the original Adult dataset, then scaled it up using the method described in [106] to create a set of larger datasets for the experiments. Table 5.5 illustrates each quasi-identifiable attribute (QID) we used in our experiments, and generalization level (GL) of each QID obtained from the taxonomy trees for Adult datasets. The "Salary" in Adult dataset is set as sensitive attributes.

5.6.2 System Environment Configurations

We configured Yarn and Hadoop Distributed File System (HDFS) using Apache Ambari. HDFS distributes data in a NameNode (worked as a master node), a secondary NameNode, and six DataNodes (worked as worker nodes). We allocated 3GB memory to Yarn

Table 5.5: Adult dataset

QID	**Distinct qid Value**	**GL**
Age	74	3
Work Class	8	4
Education	16	4
Race	16	2
Gender	2	1
Native Country	41	3

Table 5.6: Spark and dataset the configuration for all experiments

	Spark Master and nodes Parameters					Dataset and Anonymity Parameters		
	Driver Memory	Executor Memory	No of Executors	Executor Cores	No of Partitions	K Group Size	Dataset Records	Number of $QIDs$
Fig. 5.4	15 GB	3.5GB	8	3	40	50 , 250	1.2B(Billions)	6
Fig. 5.5	6.5 GB	4GB	12	3	24	50 - 1000	0.1B - 1B	6
Fig. 5.6	6.5 GB	4GB	12	3	24	1000	0.1B - 1B	2 - 6
Fig. 5.7	6.5 GB	4GB	12	3	24	1000	0.1B	6
Fig. 5.8	6.5 GB	4GB	12	3	24	10000	0.6B	6
Fig. 5.9	6.5 GB	4GB	2 - 16	3	2 - 64	1000	0.8B	6
Fig. 5.10, 5.11	6.5 GB	4GB	12	3	24	2 - 100	3.2K	6

NodeManager, and 1GB memory to ResourceManager, Driver, and Executor memories each. We used Spark version 2.1 [1] along with Yarn as a cluster manager. The details of the experimental setup and configuration are illustrated in Table 5.6 including the details of the Spark job, anonymity parameters, and dataset tuning together with configuration of CPU and Memory details.

5.6.3 Performance and Scalability

We ran experiments to understand performance, scalability, cache management of memory, and iteration management. We ran our experiments for 10 times and used the average values to ensure the reliability and consistency of the results.

5.6.3.1 Memory Effects on Performance and Scalability

In this section, we discuss our results based on three categories including: (i) comparison of the performance of our proposal against similar existing approaches, (ii) running experiments to identify the factors that ensure high performance against the growing size of records, and finally (iii) demonstrating the scalability by increasing the number of attributes in QID set and studying the impact.

We conducted the experiments to measure the performance of our proposed approach and compared it with the execution time of other approaches such as multi-dimensional top-down specialization (MDTDS) [8], multi-dimensional sensitivity-based anonymization for data (MDSBA) [9], and Apache Spark based top-down specialization (SparkTDS) [119]. In order to make sure that our approach is comparable with others, we used the

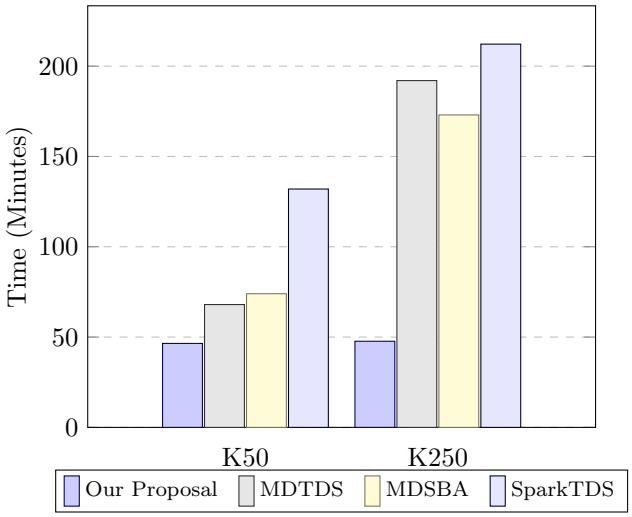


Figure 5.4: Compare the performance of proposed approach with MapReduce and Spark TDS.

same workload and enforced our Spark cluster configuration to match with experimental configuration discussed in [9] and [8] as much as possible. We also ensured that we use the same size of dataset and the size of QID for our experiments. We used 1.1GB dataset with the K-group size of K50 and K250 to compare the execution time across different approaches.

Fig. 5.4 compares the execution time of our approach with the existing approaches. However, our approach and SparkTDS are Spark-based approaches while MDTDS and MDSBA are MapReduce approaches. The results indicate that our proposal yields the lowest execution time compared to the other platforms, while SparkTDS shows the highest execution time. We also identified that our approach uses the smaller number of RDDs and effective parallelism during the execution of each partition in its respective executors. Increasing K-group size does not appear affecting the execution time significantly as our approach measures the score and update the anonymity in its prospective RDDs for all generalization level of each QID. In other words, all leaves of subtree are measured, compared, and updated for score values in two RDDs. However, the SparkTDS approach measures and updates the score of each leaf as single RDD. Thus, the increments in the generalization level will increase the number of leaves which causes more execution time for K-group size.

In addition to improved methodology of dealing with generalization level in RDD execution, there are other reasons for the performance improvement for our proposed method. First, using Spark based approach utilize efficient in-memory computing, however, inefficient memory computing [116] may leads to additional execution time as we can see in SparkTDS result. MDTDS and MDSBA approaches suffer from the underlying high latency for frequent disk access. The second reason for the performance improvement

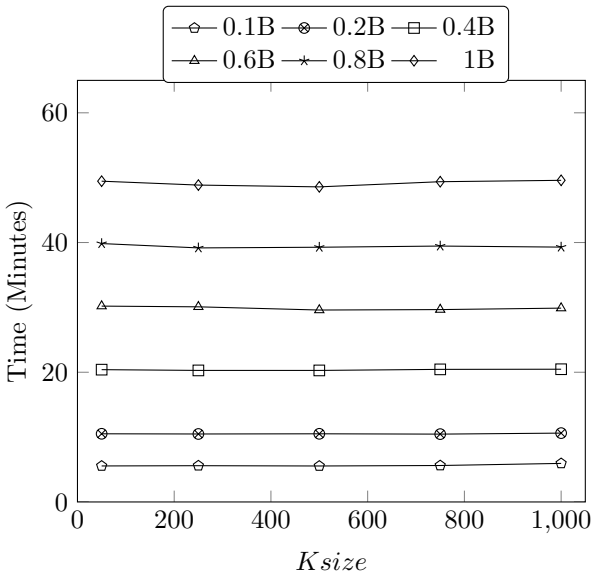


Figure 5.5: Compare the performance of increasing records number with K-group sizes

is because of the efficient partitioning which is based on the parallel execution of similar sized dataset at each node in our proposed algorithms.

We also analyzed the performance implication of our approach based on increasing the number of records sizes with respects to K-group size. We ensured that the experiments for each dataset use a constant number of partition size (i.e., 24) instead of the default partition size which vary depending on size of dataset. Fixing the partition size ensures that the data can be processed with the equal number of executors. The results in Fig. 5.5 show the execution time based on increasing records size starting from 0.1 billion (10^8 records) to 1 billion (10^9 records). We observed that the execution time has a linear growth with respect to increasing dataset size.

Fig. 5.5 also plotted the execution time against different K size ranging from 50 to 1000 records, and we did not observe any distortion caused by K-group size as the execution times remain almost constant by increased K-group sizes. We identified that this effect is because of two reasons: (i) The records are required for the measurement of privacy and utility score from RDD rather than the complete data records; thus, after each generalization step, the same records are aggregated and represented with the key value pairs. The key value pairs contain enough information and do not require additional calculation that need revisit data for calculating the score value, (ii) The anonymization process replaces the RDD original data with generalization level within the RDD by loading the score value of leaf and generalization level of QID which are broadcast to every executor of the respective nodes. This process reduces network I/O and memory, and disk I/O. Consequently, it reduces the computation time significantly.

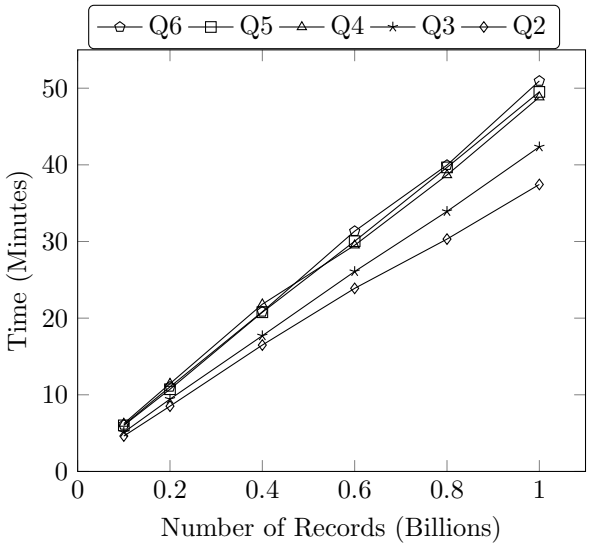


Figure 5.6: Compare the performance of increasing record size with QID numbers

In our proposed approach, we identify the K-group size by comparing the same records value with K so that if the condition is satisfied, then it terminates the algorithm for further computation.

The scalability of the distributed anonymization is benchmarked against the increasing QID size and is represented in Fig. 5.6. We increased adult dataset QID size with respect to increasing the number of record sizes. $Q6$ represents all the *qid* we used for records, whereas $Q2$ represents that two attributes are *qid* and the rest of attributes are non-sensitive attribute. We discovered that the execution time is dependent on the size of QID and variety of each *qid* value. Thus, the higher size of QID set and diverse *qid* value cause the higher execution time. Indeed, for the higher size of QID, the larger size of equivalence classes is needed to satisfy the K-anonymity requirements. It allows a greater number of attributes to be grouped/partitioned together, thus it reduces the number of required iterations.

5.6.3.2 Iteration Effects on Scalability

We analyzed the effects of iterative operation for our proposed approach with respect to increasing the number of records. In the next set of results, we identify the importance of cache for iterative intensive operations. We compared the execution time for both cached and none cached operations during the execution of anonymization process.

In the first results, we ensured that QID, partition size, and K-group size remain constant for increasing records size and number of iterations. Fig. 5.7 compares the number of iterations against the execution time for various dataset sizes. We can observe that having more iteration leads to more execution time. When we increased the record

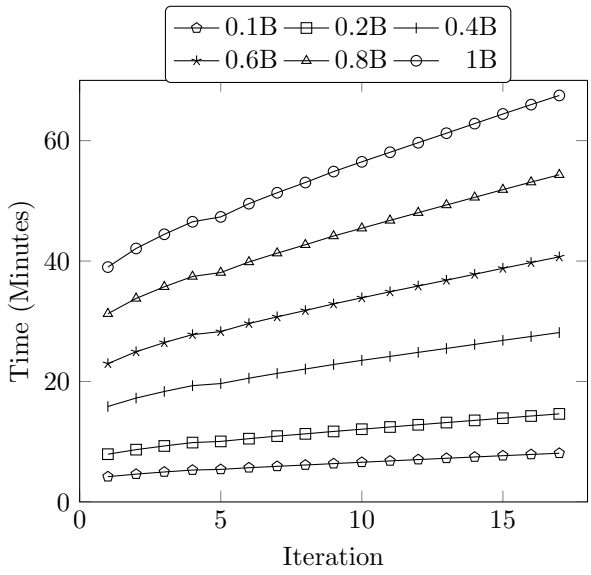


Figure 5.7: Compare the performance of increasing number of records and iterations

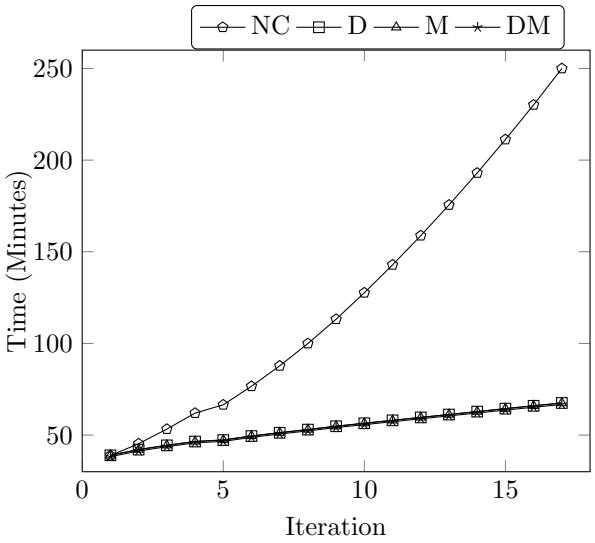


Figure 5.8: Cache and non-cache effect on increasing (iteration) generalization level

size from 0.1B and 0.2B, we observed that the executor memory had enough space to accommodate the records while processing the anonymization. Thus, it does not invoke evacuation of memory due to overload. While as we increase the records size, the executor memory starts evacuation process for the data. It is noticeable that although each RDD creates the same or less size of input data, the steps of anonymization add more data to the memory for execution. Thus, the larger the record size, the more records need to be evacuated to make enough space for execution.

Fig. 5.8 compares the cache and Non-Cache (NC) effects on increasing (iteration) generalization level. We observed that the execution time for NC RDD has higher execution time in comparison with cached RDD regardless of the storage level to DISK_ONLY (D) MEMORY_ONLY (M), or MEMORY_AND_DISK (DM). For the smaller dataset, it has more space to hold the cached RDD in memory. However, by increasing the dataset size, the RDD partitions need to be deleted from the memory and calculated again for the next transformation. In each iteration, RDD data needs to be scanned for finding the optimal generalization using privacy utility score of each *qid* attribute and TT. To achieve this, more frequent visits of RDD data in memory and storing the results into cache storage memory provides faster read and write operations while each record value is compared with K. In cache of NC iteration, the results have to be recomputed from its lineage in order to compare QID generalization and find the K-anonymized data.

While we observed that DISK, MEMORY, and their combination provide the similar execution time, all these three storage options have different approaches for accommodating cache results. As concluded by [56, 116], the combination of memory and storage are the most cost-effective operation for iteration ensuring faster computation. We also observed that after each iteration execution, the read and write time taken by memory and disk is slightly increased without recomputing the space and size for the next iteration.

Furthermore, we investigated the impact of RDD partition on the number of executors to identify a balance between high parallelism and utilizing the available resources to the maximum capacity. A partition against the executor trade-off has been discussed in [119, 144]. Having considered the results demonstrated in Fig. 5.9, we can observe that the increase in the number of partitions improves the execution time as 64 partitions (denoted as P64) has more execution time in comparison with when only two partitions are used (P2). This means that partition size and executor have to be in the balance proportion in order to avoid latency.

The overhead in data partitioning decreases runtime with increasing the number of executors as shows Fig. 5.9. For instance, the runtime sometimes increases with the number of partitions on a single worker node. However, as a general tendency, increasing number of partitions and number of worker nodes reduces the runtime significantly. Based on the results, we noticed that the runtime is slightly affected across different increasing partitions size with respect to constant K values with fixed number of nodes and partitions.

5.6.4 Privacy and Utility Trade−off

We utilized the privacy and utility metrics for the measurement and validation of our proposed method. The data anonymization technique uses trade-offs between privacy and utility to quantify the success of an anonymization algorithm. A privacy level is estimated by recognizing the uniqueness of information, a low privacy normally implies that it is anything but difficult to distinguish an individual (a characteristic, tuple or

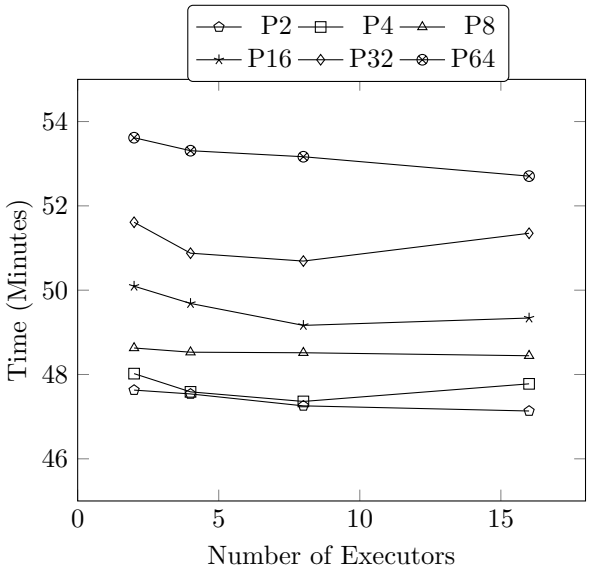


Figure 5.9: Partition(P) size impact on number of executors

record) from a group (e.g., numerous records are unique). However, a high protection demonstrates that it is (increasingly) hard to uniquely recognize a person from a group (e.g., there are numerous records having similar values). We used two privacy metrics Kullback-Leibler−divergence (KLD) and Information Entropy (I_E) to evaluate the impact of privacy level from the anonymized dataset resulted from our proposal.

In contrast, an utility level is estimated by computing the degree of degradation in the accuracy of significant value between the first value (i.e., baseline) and the anonymized value (i.e., sanitized).we use two utility metrics i.e., Discernibility Metric (DM) and Average Equivalence Class Size Metric (C_{AVE}).

5.6.4.1 Kullback-Leibler−divergence (KLD):

KLD measures the likelihood of the presence of the original attribute in the anonymized attribute for each record [76]. For example, let the original attribute of the Job is "Writer" and is anonymized into "Artist". The KLD measures the possibility of guessing the original data of "Writer" from "Artist".

In our approach, we calculate KLD on the final anonymized dataset by measuring the likelihood of the presence of each attribute and sums all the value for each attribute within a record and repeat this for all records. P_{RDD_in} indicates the sum of the likelihood of the presence of the original attribute within the original data (at a record level). P_{RDD_in} at this stage would have a very high data utility and no privacy as there is no changes made. $P_{RDD\_in(r)}$ indicates the sum of the likelihood of the presence of the original attribute within the anonymized record. P_{RDD^*} usually has lost some degree of data utility and

(and inversely has gained some degree of privacy) because the data in this set has changed from the baseline after an anonymization technique is applied.

KLD can be computed based on the formula represented in Equation (6.4).

$$KLD = \sum_{r=1}^{n} P_{RDD^*}(r) \log \frac{P_{RDD^*}(r)}{P_{RDD\_in}(r)} \tag{5.5}$$

The KLD value increases from 0 which indicates both records between the original record and the anonymized record are the same. The increase of KLD value indicates the level of privacy assurance. With the lower value of KLD, it is easy to identify the original value from the matching anonymized value (i.e., low privacy).

Fig. 5.10 presents the KLD values of our proposed subtree generalization implementation. The KLD values increase with the increase of K-group size and is very close to the comparative approaches discussed [16, 19].

5.6.4.2 Information Entropy (I_E):

I_E is used to measure the degree of how uncertain it is to identify the original value from the anonymized value within a QID set [14]. The entropy value of I_E is 1 if all the qid attributes are identical in the anonymized dataset for the same QID. To compute $I_E(QID)$, (1) the likelihood of the presence of the original attribute in a record is calculated, (2) sums up the value of (1) for each attribute in a record (denoted as $P_{RDD^*(qid)}$), (3) continues (1) and (2) for each QID, (4) sums up the value of (3) for all records. Note that if all attributes are changed between the original record and the anonymized record, the value of P_{RDD^*} is 1.

$$I_E = -\sum_{qid=1}^{n} P_{RDD^*(qid)} \log P_{RDD^*(qid)} \tag{5.6}$$

Based on the Equation (5.6), we computed $I_E(QID)$ for single QID. To obtain the I_E for the entire anonymized dataset (denoted as RDD^*), we calculated the I_E for RDD^* by taking the average of all QID. The entropy value of I_E is 0 if there are two identical records from the original dataset to the anonymized dataset for a matching equivalent class. The maximum value of I_E is achieved when the original record sets is very different from the anonymized record sets for a given QID. Higher value of I_E represents more uncertainty (i.e., higher privacy).

Fig. 5.10 shows the privacy level in terms of entropy of our proposed approach. Generally, the entropy increases with K size. Though the I_E score is highest at ($log_2 k$) in [99], the I_E score of our proposal works better than the scheme proposed by [99]. The performance of our approach is close to more high entropy standards and can achieve much higher privacy levels which are similar to the scheme proposed by [19]. As the entropy represents the information content of a data change, the entropy after data anonymization

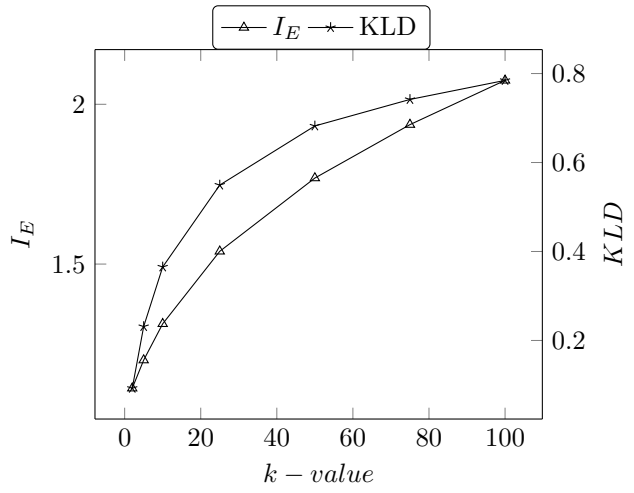


Figure 5.10: Kullback-Leibler−divergence and Information Entropy scores of our approach

should be higher than the entropy before the anonymization which is the phenomenon observed in Fig. 5.10.

5.6.4.3 Discernibility Metric (DM):

DM reports the data quality resulting from the degree of data degradation, as a result of data anonymization, of an individual' tuple based on an equivalent class (EC). Let EC be the set of equivalence classes of a K-anonymized Table. EC_i is one of the equivalence classes of $\mid EC \mid$. The DM metric can be expressed more formally for RDD^* as Equation (5.7).

$$DM_{score} = \sum_{EC_i \in RDD^*} \mid EC_i \mid^2, \tag{5.7}$$

where i represents a qid_{tuple} within an equivalent class. The data utility is associated with the DM score. If DM score is high, it means the data utility is low (i.e., the original qid_{tuple} has lost its original values) while the lower the DM score represents the data utility is high.

Discernability Metrics (DM) [29] measures the cardinality (i.e., distinctness) of the equivalence class. For a low value of K, the cardinality of equivalence is too small. If the privacy level is high (high value of K), the discernability metric increases sharply which increases the cardinality of an equivalence class. Equivalence classes with large cardinality tend to group datasets in a large range leading to large information loss. Fig. 5.11 presents the discernability penalty of 32000 records.

We observed that the overall trends for the DM to the DM values observed in other similar approaches in [19, 99]. The increases K-group size increase the EC records, thus decreases the cardinality from each other. We observe the trends remain steady for the

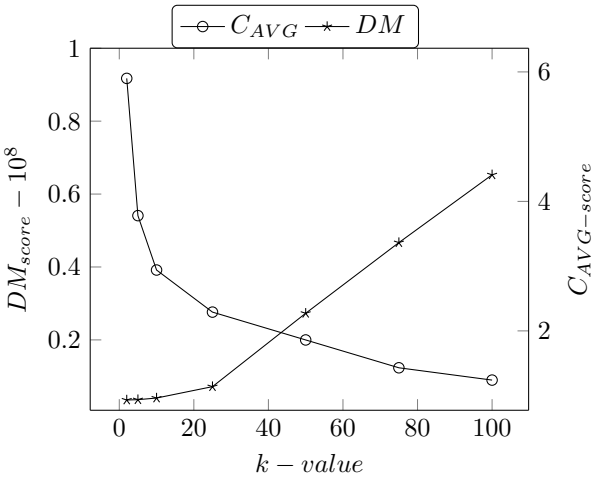


Figure 5.11: Discernability and Average Equivalence Class Size Metric scores of our approach

Adult dataset, as a result a low sensitivity effect to the increasing growth of K-group size.

5.6.4.4 Average Equivalence Class Size Metric (C_{AVG}):

C_{AVG} is used to measure data utility based on attributes of the average size of the equivalence class. The increase in the number of equivalence sizes results in the higher data utility as it is more difficult to identify an attribute among many identical attributes. In K-anonymized dataset, the size of the equivalence classes is greater than or equal to K. As a result, the quality of the data is lower if the size of all or part of the equivalence classes greatly exceeds the value K. The score of C_{AVG} is sensitive to the K-group size [81]. C_{AVG} for RDD^* is calculated as the Equation (5.8).

$$C_{AVG} = \frac{|RDD^*|}{|EC|}/k \tag{5.8}$$

The total number of records of RDD^* is donated as $|RDD^*|$, whereas $|EC|$ represents the total number of equivalence classes.

Fig. 5.11 represents the results of C_{AVE} for increasing group size of K. The decreasing score value against the increasing size of K is observed indicating that the size of the created ECs is equal to the given K, that is the ECs contain the number of generalized records that satisfy the K-anonymity. As the value of K increases, the EC has more records than the K requirement due to higher generalization level, this keep increasing the C_{AVE} score value.

5.7 Conclusion and Future work

This study addresses the design consideration and proposes a generic framework for implementing subtree-based generations on Apache Spark. Our proposal is implemented using a series of RDDs that supports more efficient partition management, improves memory usage, and supports better iteration process which is much more suited for an iteration-intensive algorithms such as subtree generalization. Our proposed approach not only reduces the complexity of operation and improves the performance, but it also shows high data utility scores while maintaining competitive level of data privacy required for any data anonymization techniques. We plan to extend our study by further exploring the suitability of other data anonymization approaches for Apache Spark platform. For instance, we plan to investigate one of the multi-dimensional data anonymization strategies such as Mondrian [81] to examine the support for recursive operations in Apache Spark.

Chapter 6

A Novel Hybrid Approach for Multi-dimensional Data Anonymization for Apache Spark

Summary

Multi-dimensional data anonymization approaches provides more fine-grained data privacy by allowing a different anonymization strategy applied for each attribute. Many variations of multi-dimensional anonymization (e.g. Mondrian) have been implemented in different distributed processing platforms (e.g., MapReduce, Spark) to take advantage of their scalability and parallelism supports. However, the existing implementation methods, either iteration-based or recursion-based approaches, often suffer from many overheads due to lack of support from the platforms for running iterative intensive tasks with multi-dimensional features. In this study, we propose a novel hybrid approach for multi-dimensional data anonymization strategy based on Mondrian. Our hybrid approach creates a lot fewer numbers of RDDs and the smaller size partitions attached to each RDD compare to existing approaches. These contribute towards reducing many overheads involved in re-computation, shuffle operations, message exchange and cache management. The experiment results illustrate that our proposal outperforms other similar methods in terms of performance and scalability while maintaining high levels of data privacy and utility.

6.1 Introduction

The multi-dimensional based data anonymization approach (e.g., Mondrian [81]) provides higher data utility compare to single dimension-based data anonymization (e.g., subtree generalization [50, 136]) as it allows more fine-grained generalization for each attribute [6, 46, 150].

The MapReduce [40] based approaches have been proposed to implement various multi-dimensional [81, 82, 96] data anonymizations to provide scalability solutions that can work well with big data. By and large, the existing proposals can be divided into two categories: recursion base approach and iteration-based approach. The recursion based approach typically involves a multiple MapReduce jobs in which each involves: (1) dividing records to each worker node as subset, (2) calculating the split point (i.e., also referred as "cut") in for each subset, (3) and joining all the split points of subsets then measures a single split point from the joined records from the subsets. These main MapReduce jobs are repeated until full anonymization across different features (i.e., attributes) are met as seen in [29] and [155]. On the contrary, the iteration-based approach utilises data partitions by using a single MapReduce job at each iteration. The MapReduce job is responsible for data division and performs computation required for anonymization on each iteration. The result of the computation for each iteration is stored and used for a subsequent iteration as an input [154]. However, these approaches suffer from inherent MapReduce limitations that include expensive disk based I/O and sequential execution of jobs [19, 24, 71, 72, 116].

To address these MapReduce related concerns, Spark has been proposed [144] with the support for in-memory computation, data transformation across multiple execution nodes, as well as the options for a several memory strategies [143]. Many data anonymization approaches which often require intensive repetitions have been implemented to take the advantage of Spark's new advanced features. The majority of existing Spark based data anonymization solutions utilize the default iteration support for single-dimensional data anonymization strategies [8, 19, 28, 52, 119]. The papers [11, 107] proposed Spark based iterative approach for Mondrian with some significant overheads in coordinating iterations across multiple dimensions. Recursive operation-based data anonymization approaches have been proposed with some extensions [59, 74, 80]. For example, the records in the nested RDDs require multiple extra operations on the same RDD as Spark does not support nested RDD operations [74]. To remedy this problem, recursive approaches were redesigned and implemented in Spark using iteration support [109].

In this chapter, we propose a novel hybrid approach for multi-dimensional data anonymization strategy based on Mondrian. Our hybrid approach creates a fewer number of RDDs compare to the number produced by recursive-based approach while the size of RDD partitions are much smaller than the partition size created by iteration-based approach. The reduced size of partition along with far fewer number of RDDs created by our data anonymization for multi-dimensional data significantly reduces many overheads. The main contributions of our hybrid approach are as followed:

- We illustrate two different approaches can be implemented for Mondrian on Spark and provide in depth discussion on the strengths and weaknesses of the two approaches.

- We propose a novel hybrid method which creates far fewer number of RDDs with much smaller size of partitions associated with each RDD. We demonstrate that our method is especially effective when a data anonymization strategy is involved to process multi-dimensional features such as seen in Mondrian algorithm
- We highlight the improvement in execution complexity as well as the key factors that contribute towards reducing many execution overheads associated with re-computation, shuffle, massage exchange and caching management in spark.
- We provide experimental results to demonstrate that our approach outperforms other similar methods and can be used as a scalable, high-performance, and privacy-preserving data anonymization solution for big data.

The rest of this chapter is organized as follows. Section 6.2 provides the related studies to our work. Section 6.3 provides the background knowledge required to understand Mondrian algorithm with an example. Section 6.4 describes two potential approaches to implement Mondrian on Spark along with the discussion of the challenges and limitations of each approach. Section 6.5 describes our proposal from the points of design principle, algorithm and how our proposal mitigates many overheads. Section 6.6 describes the details of experimental results and key findings. Section 6.7 provides the conclusion and future work.

6.2 Related Work

The original papers on Mondrian algorithm suggest two different strategies for dimension selection and dividing data within a selected dimension: one is based on the normalized values [81] and the other based on the computation of workload [82]. With the explosion of dimensions involved in big data, the serial execution, typically associated in the earlier implementation strategies of the Mondrian, was found to be no longer applicable as the strategy quickly escalates into creating insufficient memory [97, 155].

More scalable strategies to implement Mondrian on MapReduce have been proposed [29, 154, 155]. Zhang et al. [155] provides an approach by using series of MapReduce jobs designed to perform recursive computation. To find the split point, the authors provide the median of medians approach in which a several MapReduce jobs are used to split data into partitions then measures a median value in every Partition. The median values of all partitions are combined followed by a MapReduce job to measure the median of the combined results to find a new split point. In their approach, the recursive MRMondrain uses MapReduce jobs to validate the anonymity and repeats the process recursively to achieve a desired anonymized result. Another MapReduce based recursive approach was proposed [29]. In their approach, the MapReduce jobs are assigned to a single reducer for a single equivalent class and calculates the count for each attribute in the first phase.

In the second phase, it splits the data into the left and right partitions iteratively using the midpoint value calculated in the first phase for each attribute. In [154], it proposes an iterative approach to reduce computational cost associated with the earlier recursive based operations in MapReduce. In their approach, the iterative MRMondrian uses the partition ID index tree to perform a recursive step iteratively. A MapReduce job divides each partition small enough to fit it in the memory of each cluster node then runs multiple partitions in parallel on each node to execute the Mondrian algorithm.

Spark based solutions have been proposed further to address the limitations associated with MapReduce based implementations [64,87]. Anonylitics [107] utilizes Spark's default iteration support to implement the Mondrian. However, their approach brought additional challenges because increasing number of iterations demand for increasing memory space. Because of the lazy evaluation principle in Spark where the actual execution of Spark jobs does not start until RDD transformation is triggered, the intermediate data created by each iteration requires to be stored in the memory which eventually may exhaust the memory space allocated for each RDD. PRIMA [11] implements the Mondrian algorithm with Optimal Lattice Anonymization(OLA) in their anonymization strategy. The OLA is used to define the utility and generalization level rules in order to limit the data utility loss. The paper [97] propose a distributed Mondrian approach to split the input data to each node of cluster by using Spark k-mean. Partitions then are assigned to the nodes in each cluster. A series of Spark jobs runs on each cluster node to produce anonymized results which are combined later by another cluster node. An iteration-based approach for data anonymization in Spark has been proposed [19,28]. However, these proposals focuses on data anonymization strategies on a single dimensional data without consideration for the input samples with multi-dimensional features, therefore, these existing methods are difficult to apply for Mondrian. A recursive function-based data anonymization in Spark has been proposed by [11]. However, their approach suffers from many limitations such as repeated operations on the completed dataset that already satisfies the k-anonymity constraints which wastes many resources (e.g., memory) and high network communication overhead as criticized by [116].

6.3 Background

6.3.1 Mondrian Algorithm

LeFevre et al. [81] presented the original Mondrian algorithm where it splits the original dataset and applies a K-anonymity based anonymization strategy until the K-anonymity constraint meets. In his approach, it first identifies a dimension (i.e., attribute) and finds the value that has the highest occurrences in the dimension. The Median value of the dimension is measured using the frequency set and it is used to divide the data into two subsets. It performs the same operation on the subset data in a recursive manner in which each split of the subset carries the first split information all the way to the

last split until the anonymization condition is satisfied [77]. All the split values (i.e., the results of recursions) are joined together to ensure data is anonymized. Each partition in a recursion may create a lineage tree (cut) starting from the topmost generalization level to the lowest level of generalization [82]. The resulting anonymized data in different multi-dimensions usually contains the data with better quality than the data anonymized in a single-dimensional generalization (e.g., subtree generalization [50, 136]). Algorithm 14 depicts the main steps involved in the Mondrian.

Algorithm 14: Multi-dimensional Mondrian [81]

```
   Input: Partition ← Dataset
   Output: Anonymized Dataset ← Summary
 1 Anonymize(Partition);
 2 if no allowable multi-dimensional cut for Partition then
 3 |   Summary ← Partition        /* Partition is anonymized              */
 4 return Summary
 5 else
 6 |   dim ← Select_dimension()      /* Allowable multi-dimensional cut     */
 7 |   F_s ← Frequency_set(Partition,dim)
 8 |   SplitPoint ← median(F_s)
 9 |   LHS ← {R ∈Partition:≤SplitPoint}
10 |   RHS ← {R ∈Partition:>SplitPoint}
11 return Anonymize(LHS) , Anonymize(RHS)
```

The initial step of the algorithm considers all data as a single partition and performs a validation for K anonymity condition. A dimension (QID) is selected for further partitioning as long as the K constraint has not met. To choose a dimension for a partition, Mondrian uses an attribute from the QID with the highest generalization level. If multiple dimensions (QIDs) have the same generalization levels, the first one that validates as "an allowable cut" is selected. Once a dimension (QID) is selected, Mondrian uses a median partitioning approach [81] to choose the split value. In this approach, the frequencies of distinct values for each attribute within the (QID) is calculated. The attribute with the median value across the frequency sets is decided as the spilt value.

Example:

Consider the dataset in Fig.6.1 where there are quasi-identifier attributes such as 'Age', 'Gender', 'Education' and sensitive attribute 'Income' as private information.

The dataset starts with one dimension(e.g., 'Education') and moves to the next dimension to find more allowable cuts. An allowable cut consist of records numbers in a region equal to or more than K value. Whereas none allowable cut contains the record number in a region less than K value, The none allow able cuts contains anonymized records and stop the partitions formations.

For example, let's consider it is a K=2 anonymisation. The first allowable cut for the dimension 'Education' is in between "Master" and "Doctorate", which divides the dataset into two groups, one having 4 data points while the other has 2 data points, as

Age	Gender	Education	Income
35	Female	Master	75000
35	Male	Doctorate	55000
36	Female	Master	82000
37	Female	Bachelors	28000
37	Male	Doctorate	48000
38	Female	Master	90000

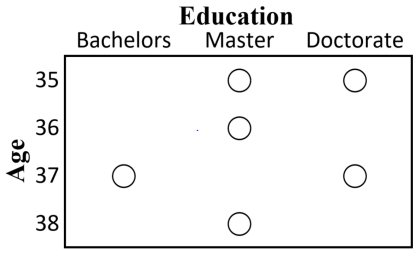


Figure 6.1: Example Adult Dataset and its Spatial Representation

Age	Gender	Education	Income
35-38	Female	ProfSchool	75000
35-38	Male	Doctorate	55000
35-38	Female	ProfSchool	82000
35-38	Female	ProfSchool	28000
35-38	Male	Doctorate	48000
35-38	Female	ProfSchool	90000

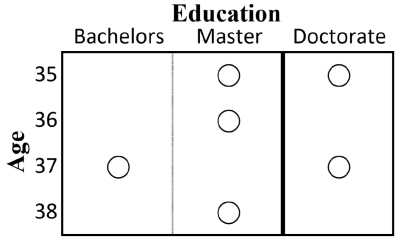


Figure 6.2: Adult Dataset Using Single Dimensional Anonymization its Spatial Representation

seen by the solid line in Fig.6.2. This cut is allowable because the number of records in both groups are greater than or equal to 2 (i.e., K group size). The cut in between "Bachelors" and "Master" is none allowable cut. To obtain the allowable cut the already divided dataset is further split into two more subset as left and right subset, we notices left side of the dashed line has only one data record that satisfy specified K anonymity value, therefore, K anonymity condition remaint not satisfied. Fig.6.3 further divides the dataset for another dimension 'Age' attribute on top of the dimension 'Education', hence multi-dimensions. Another allowable cut exists in between the age 36 and 37, both groups having 2 data points in which still does not violate K=2 constraint. As it can be seen in Fig.6.3, the distribution of data points is less distorted and more generalized when compared to a single dimension cut shown in Fig.6.2.

Age	Gender	Education	Income
35-36	Female	Master	75000
35-37	Male	Doctorate	55000
35-36	Female	Master	82000
37-38	Female	ProfSchool	28000
35-37	Male	Doctorate	48000
35-38	Female	ProfSchool	90000

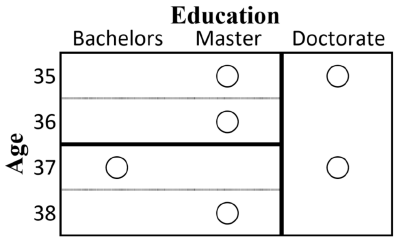


Figure 6.3: Adult Dataset Using Multi-Dimensional Anonymization its Spatial Representation

6.3.2 Spark Architecture

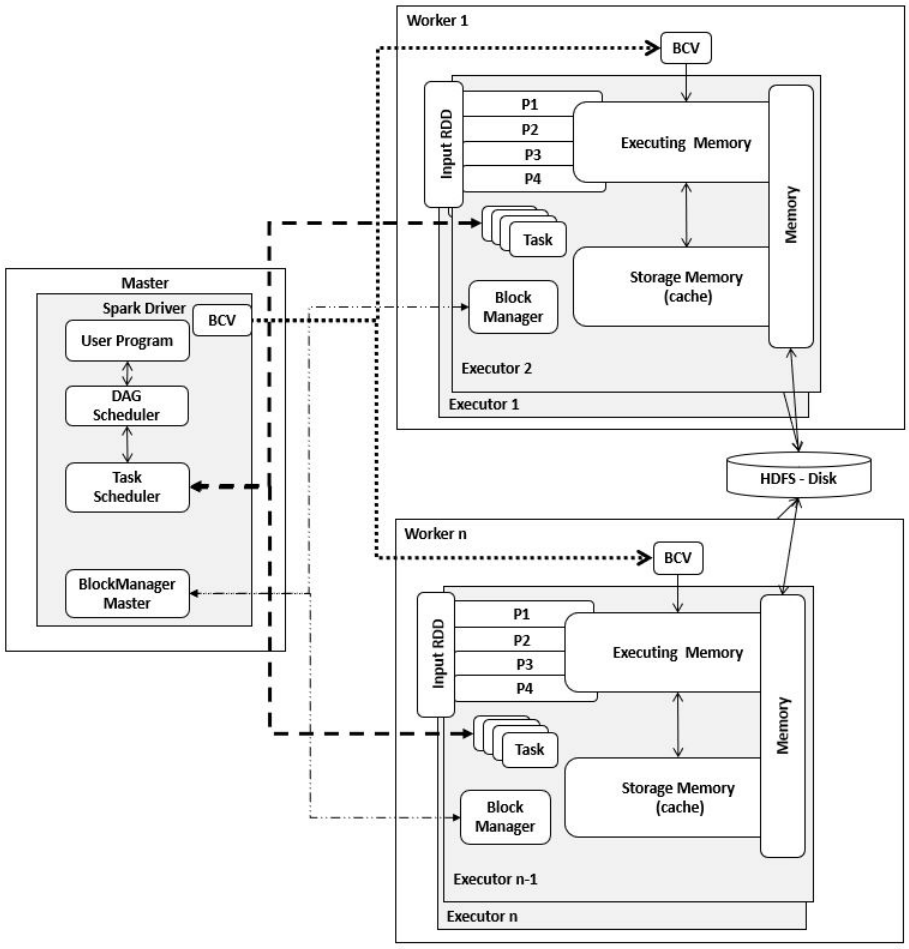
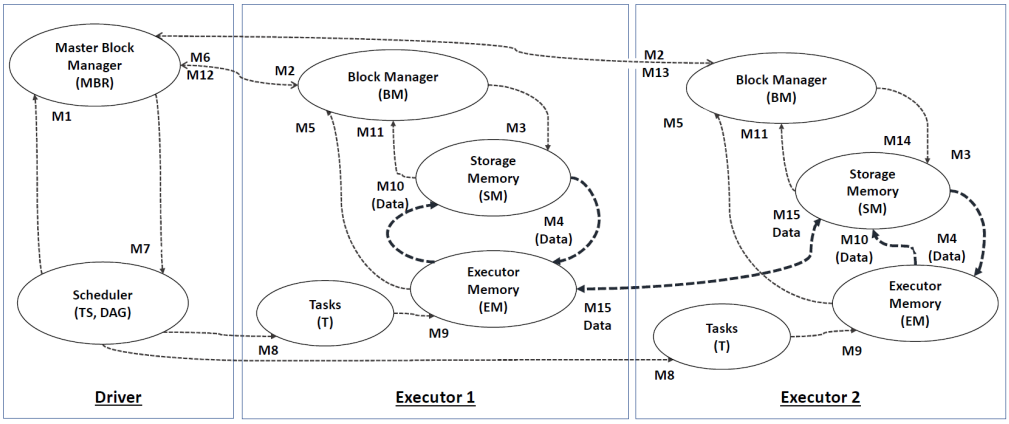


Figure 6.4: Spark Driver and Executor

Resilient Distributed Dataset (RDD) is the main building block of Spark architecture. RDD is a read-only partitioned collection of objects (e.g., records) where each partition can be represented in the memory or disk. RDDs can only be created through deterministic operations in three different ways: by loading an external dataset from a stable storage (e.g., HDFS), by the operation on existing RDDs, and by distributing a collection of objects to the executing nodes across networks by RDD transformations.

The data stored in a RDD is split into partitions in which each of them contains a subset of the data. These partitions are distributed over the nodes across different clusters while each node can host one or more partitions. The creation of a new RDD may transfer the data in different worker nodes as the result of shuffle. A shuffle operation can move data from one node to another node across networks. The memory of an executor is dynamically divided into the execution memory and the storage memory for processing



M1: Driver TS seeks for Input RDD location in MBR,
M2: Driver MBR communicate to executor's BM for Input RDD updates,
M3: Each Executor BM seeks input RDD partitions in the SM,
M4: Executor SM loads the partitions in EM by transferring the data,
M5: EM updates Executor BM the input RDD statues,
M6: All Executors BM update Drivers MBR for Input RDD locations,
M7: Drivers MBR updates Driver TS and DAG for data locations and size,
M8: Driver TS assign the task to each executor according to available recourses to perform Map operation,
M9: All Executors assign a task to perform execution on Executor EM data,
M10: The results of each task are move to SM of Executors to next execution,
M11: Updates Executors BM about the data movement,
M12: Same as M6 followed by M7, M8, and M9 for next Task Execution to perform Reduce operation,
M13: MBR communicates with Executors 2 for Partition exchange for Reduce operation to load in memory,
M14: Same as M3 followed by M4 for all executors,
M15: Exchange the records in partitions followed by M10, M11, M12, and M7 accordingly for both executors.

Figure 6.5: Spark Driver and Executors, data and message exchange during execution

and holding RDD partitions during the execution. The execution memory is used to process the data assigned for the RDD while the storage memory is used to store the intermediate data created as the result of processing the RDD. In a RDD transformation, Spark driver assigns a single task to a partition where a task represents a unit of execution. The increase in the number of partitions requires Spark to increase the number of tasks accordingly. The execution of tasks across the number of partitions decides the degree of parallelism in the cluster [35].

An Apache Spark cluster provides computations services using master and multiple worker nodes. The master node host driver for task scheduling and dispatching task to executor in worker nodes. Spark use multiple executors to access data from workers physical data storage and perform computation using worker recourse on the data. The driver coordinates the processes that are running on the worker nodes and distributes the application code to the worker nodes, launches executors, and then assigns tasks to the executors. Executors are created in worker nodes on request from the master and generally live for the entire application unless they crash or are killed prematurely by the

master. If an executor crashes, another one is immediately created to replace it. The driver keeps track of the operations and outputs generated by each RDD to avoid unnecessary recomputation. We show the structure of Spark with the driver and executors in Fig.6.5. The M4 and M10 transfer the data within the executor often between memory to memory and sometimes memory to disk. M15 is considered to be the most expensive communication as it transfers data from one executor to another executor using a shuffle operation while the rest of massages are required for information exchange between the driver and executors.

6.4 Mondrian Approaches in Spark

We provide the descriptions of two different approaches that can be employed for implementing Mondrian on Spark platform: recursion-based and iteration-based, respectively. We discuss the strengths, weaknesses and implications of these different approaches.

6.4.1 Recursion Based Approach

In this approach, data is split into a number of subsets continuously until each subset meets K-anonymity constraint [147]. We provide the recursion-based approach with an example in Fig. 6.6. The recursion approach checks for the complete dataset represented as RDD1 for the K-anonymity constraint. Until this constraint is violated, the generalization based on a taxonomy tree is applied before dividing data into two parts for the selected dimension. For a selected dimension, it checks the frequency of attributes and identifies the split point using the median approach [155]. The median point of RDD1 is required to sort and shuffle each record. The median point is used to perform two operations for splitting data RDD1 [95, 143, 144]. The first set of records above the median point are assigned to left (RDD2) while the rest records equal or less than median points are assigned to right (RDD3).

The further spilt (cut) follows for the left (RDD2) by further creating the left (RDD4) and the right (RDD5) while the right (RDD3) is divided into the left (RDD6) and right (RDD7). The growth of RDDs continues until it reaches the point where the records in each RDD are less than K=2 group size. In the last created RDD, the records are anonymized and all child RDDs are joined together.

Creating a child RDD also creates many partitions at each round of recursive operation. The size of newly created partitions at each child RDD is usually smaller than the size of the parent partition [109,117,142]. The total number partitions created by all child RDDs does not exceed the number of partitions allocated to the parent RDD. For example, a 10GB parent RDD with 100 partitions can be divided into two child RDDs with 100 partitions each. However, the size of child RDD changes to 5GB each. The number of partitions determines the size of tasks that can be parallelized in the algorithm, which affects the execution efficiency. Typically, having more RDDs (i.e., consequently creating

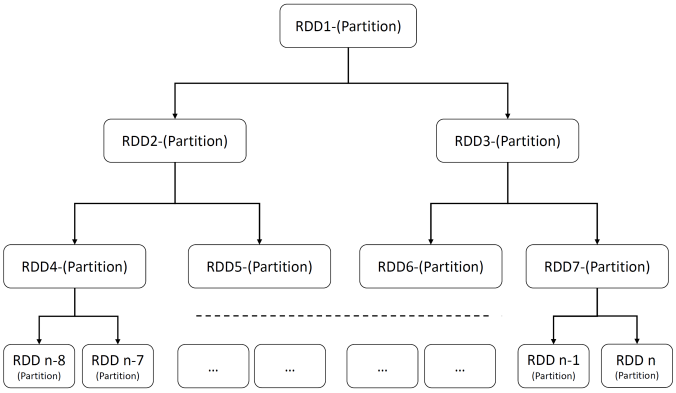


Figure 6.6: Recursive Mondrian

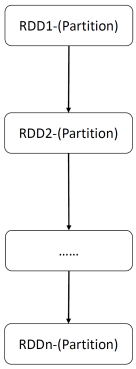


Figure 6.7: Iterative Mondrian

more partitions also) can create overheads due to more message exchanges involved across different partitions of different RDDs.

6.4.2 Iteration Based Approach

In the iteration-based approach, each round of iteration validates the K-anonymity constraint and applies a generalization level to one dimension at a time. Fig. 6.7 depicts the iteration-based approach in terms of RDDs. The input records (RDD1) are first checked for the K-anonymity constraint in the first dimension. If the first dimension does not satisfy anonymity, it applies a generalization to the dimension followed by increasing the generalization level and stores the results to RDD2. If the first dimension satisfies the anonymity requirement, it checks the requirement in the next dimension and repeats the process for all dimensions. In this approach, all the dimensions are computed progressively and apply a generalization at each iteration to the respective dimension, hence no median point is required to divide the data into multiple RDDs. However, the validation of anonymity must be performed starting from the first dimension in each round of iteration to the last [104].

In this approach, the growth of RDDs is depended on the number of generalization levels, K-anonymity, and the number of dimensions. Each iterative operation contains one RDD execution according to the parent RDD to produce the results for the child RDD. The data size and the number of partitions for all child RDDs are the same as to their respective parent RDDs. It notes that the initial and final iteration have different processing time. The first iteration only validates the k anonymity condition for the initial QID and finding a dimension while the final iteration validates the k anonymity condition for all and every QIDs and applies the generalization for the last selected dimension.

6.4.3 Challenges and Limitations

RDD transformation typically forces Spark to create children RDDs rather than using the same parent RDD. The increase in the number of new RDDs adds computation cost challenges. We identify the following Spark components that are affected due to the increasing number of RDDs as the results of both approaches and how they add significant overheads to various aspects of executions.

We represent the number of growing RDDs in recursion and iteration using following equations.

$$Recursion = \sum_{j=0}^{i} 2^j \cdot \frac{n}{2^j \cdot (k * GL)} \tag{6.1}$$

$$Iteration = \frac{n}{(k * GL)} \tag{6.2}$$

Where n represents the number of records, j represents the total number of split points whereas i is used as an incremental counter for measuring the depth of recursions. Where GL denotes generalization level in K-group.

6.4.3.1 Re-Computation cost

Spark execution memory and storage memory plays a very crucial role in the execution of recursive and iterative operation tasks. During the execution, partitions are loaded from storage memory to executing memory as shown in Fig. 6.5, as indicated by the messages M4 and M10. The results of parent partitions are moved to the storage memory of the child partition by removing existing partitions from the storage memory using Least Recent Used (LRU) approach [144]. The next child partitions may require the removed partitions from the storage memory for the subsequent process. The re-computation requires more time to recreate the deleted partitions in the storage memory which results in increasing execution cost.

Recursion Based Mondrian creates two children RDDs as a result of an operation on a parent RDD at each round of recursion. It uses execution and storage memory intensively to hold child RDD partitions. For example, the execution memory loads the RDD1 partitions after processing the input data. Then, it updates the storage memory with two children's RDDs (RDD2 and RDD3) to have the same number of partition numbers as their parents. Thus, at each round of execution, according to Fig.6.6, each child RDD splits into further two RDDs. Although the data size of child RDD at each round of recursion gets smaller and smaller, the same number of partitions requires the same space for each child in execution and storage memory as parent [143]. The increased number of RDDs increases the eviction of child/parent RDD required for subsequent computations and increases the delay in execution for re-computation of the partitions of the deleted RDDs.

Iteration Based Mondrian processes a parent RDD transformation and creates a single child RDD. In each iteration round, the child RDD inherits the results of the data

that are generalized for selected dimensions from its parent RDD. The child RDD saves this data in memory for further processing. It notes that the size of child and parent RDD remains the same throughout the iterative execution. After the execution, the partition of the parent RDD is evicted from the storage memory. Though the execution memory only requires for the most recently created RDD partitions, however, because of the limited capacity of storage memory, Least Recent Used (LRU) can be triggered to make space for new partitions [65].

6.4.3.2 Additional Shuffle Operations

A shuffle process involves moving data from an executor over the network to the other executor as depicted in Fig. 6.5, indicated by M5. The shuffle process mainly depends on network transfer capability between nodes thus any network delay adds additional shuffle overhead.

Recursion Based Mondrian triggers shuffle for computation of the median point (split point) and joining all the records in the final step to get anonymized RDD. In this approach, child's RDD partitions are larger in number but smaller in data size. The shuffle cost of moving data gets smaller and smaller at each round due to a smaller number of records (i.e., smaller data size) in each partition. However due to the increasing number of partitions at each round, the shuffle cost actually increases [36].

Iteration Based Mondrian triggers shuffle operations for each round of iteration as it requires to update the record with a generalization level in each iteration as well as to compare the records for K group. The shuffle cost in this approach is smaller compare to recursion approach. This is because the shuffle operation in this approach only moves non-generalized records for validating k-anonymity constraint. In addition, the later stages of iterations require a shuffle process to move a smaller size of data compare to the early stages.

6.4.3.3 Increase in Message exchange

Spark driver continuously communicates with executors across different nodes to get up to date information as to what operations were performed as well as what results were produced by each executor. The driver ensures at least one task is assigned to one partition. The statues of results of each task performed by an executor are communicated back to the driver [93] [98]. The executor keeps track of the locations of all RDD partitions and updates any change of partition locations made by block manager. Any changes in the block manager is communicated back to the driver to update the master block manager. These are shown in Fig. 6.5, as indicated by the messages M2, M6, M8, M12, and M13. The driver exchanges these information with other executors that are processing the same RDD partitions. The increase in the number of RDDs inevitably increase the frequency of these communications [116].

Table 6.1: Various overheads

	Re-computation Cost	**Shuffle Overhead**	**Message Exchange**	**Excessive Caching**
Recursive	High to Higher	High to Higher	High to Higher	High to Higher
Iteration	High to Medium	High to Medium	High to Medium	Medium

Recursion Based Mondrian increases the communication cost as the number of children RDD increases. The operations on each child partitions need to be communicated back to the driver which inevitably increases communication overhead. The communication overhead worsens according to the number of children RDDs and associated partitions are created as more recursions occur [74].

Iteration Based Mondrian requires a smaller number of message exchange compare to the recursion approach due to limited number of newly created RDDs. However, each iteration updates the driver for newly created child RDD partitions in each executor. As the number of data size and partitions are the same, the iteration message exchange overhead remains the same for all subsequent iterations [134].

6.4.3.4 Excessive Caching

The increasing number of RDDs also increases the requirement for having to cache the data generated by both the parent and children RDDs for further referencing purposes. The disk storage may be used along with memory (Storage memory) to cache the RDD. The RDD which holds more data is mostly required to be cached in disk due to insufficient memory, an increase in cache increases the execution overhead due to disk I/O [57].

Recursion Based Mondrian holds every parent RDD in the cache as the information of the parent RDD is required to create children RDDs. The disk-based cache that holds RDD partitions are accessed by the executor for computation of new child RDD. The growth of child RDD requires caching the results for the subsequent process as well. The child RDD that are not required to be processed further during the execution is cached in the disk due to insufficient memory. Once the recursive execution is complete, all the cached children RDDs are loaded to the executor for a join process. Although the data size of each child partition keeps reducing before the join process, however, the increasing number of RDD partitions requires more interaction with disk which increases significant I/O Overhead due to cache [59].

Iteration Based Mondrian uses disk as a cache for each round of iteration as the storage memory is allocated to the parent RDD partition which must be discarded and make space for new child RDD partitions. Processing and accommodating large data in disk cache requires more overheads compared to memory which can slow down overall computation [145].

The overview of various overheads associated with different aspects of both approaches are summarised as following in Table 6.1.

6.5 Proposed Solutions

We propose a hybrid approach to implement Mondrian algorithm most effectively on Spark. We describe the design principles, new Spark RDD-based algorithm, and address how our proposal mitigates the various overheads we discussed earlier.

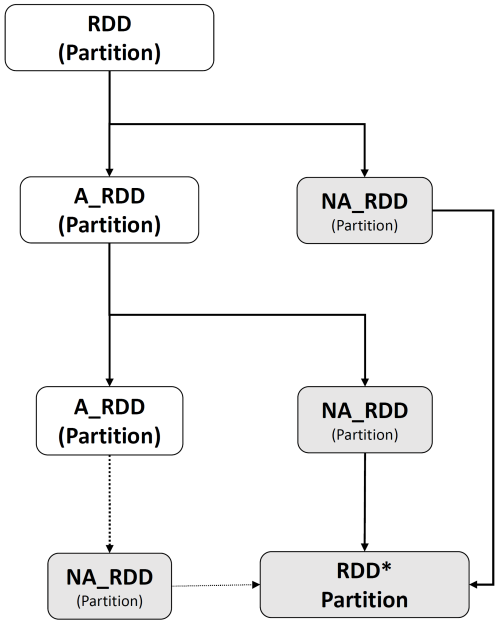


Figure 6.8: Flowchart of RDDs in our hybrid approach

6.5.1 Design

Recursive operations in our approach is limited to be mainly used for partitioning data that do not require any further cuts. This is to control the creation of unnecessary RDDs. Iteration operations are mainly used for further cuts on multi-dimensional data and validation for K-anonymity constraint. The overview of our approach is described in Fig. 6.8 where NA_{RDD} mainly performs recursive-based operations while A_{RDD} performs iteration-based operations. The grey shaded boxes represent NA_{RDD} which operates on a small number of partitions - we call them "Passive RDD" as these RDDs are only required in the final stage. The records that do not require further generalizations are filtered as anonymized record to create Non-allowable Cut RDDs (NA_{RDD}). The rest of the records are kept in Allowable Cut RDDs (A_{RDD}) after a generalization is applied. To avoid creating further smaller sized partitions for children RDDs, we enforce the re-partition approach which collects all RDD partitions then makes new partitions according to the size of RDDs. This ensures the size of the partition in newly created children RDDs does not exceed the size of its parent RDD. The partition numbers and

data size of A_{RDD} partition keep getting smaller and smaller compare to the input data RDD. We call them "Active RDD" as these RDDs are required for subsequent stages.

6.5.1.1 Algorithm

Let $IRDD$ be an input dataset which contains the records r. Where $r = (QID_{tuple}, SA)$, QID_{tuple} is the tuple of given QID also known as a dimension dim while SA indicates sensitive attributes. Each QID contains C_{qid} where the P_{qid} of the respective C_{qid} is stored in a taxonomy tree TT with generalization levels. K represents an anonymity constraint (i.e., K-group size) that is used to validate if an anonymized set satisfies the constraint. We present a dimension where generalization has all been applied as $dim*$ which to result in QID^*_{tuple}. A_{RDD} represents allowable cuts that require further anonymization whereas NA_{RDD} is Non-Allowable Cuts representing the anonymized records r which to be ready to produce the final anonymized dataset RDD^*. We illustrate our proposed solution in Algorithm 15.

Algorithm 15: Generalization Phase of Spark subtree.

```
   Input: RDD, k
   Output: RDD*
 1 Anonymized(RDD, k);
 2 r ← RDD;
 3 for i = 1 of n in r do
 4     QID_tuple, SA ← r;
 5     if QID_tuple satisfied k then
 6         r* ← QID_tuple, SA ;
 7         NA_RDD ← r*;
 8         NA*_RDD ← cache(NA_RDD, Re_partition);
 9         return NA*_RDD;
10     else
11         for dim of n in QID_tuple do
12             |EC| ← dim (EC) ;
13             if |EC| ≤ k then
14                 dim* ← dim ;
                   /* Replace C_qid ← P_qid from TT                    */
15                 QID*_tuple ← dim*;
16                 A_RDD ← QID*_tuple, SA ;
17                 RDD ← (A_RDD, Re_partition) ;
18                 returnAnonymized (RDD,k);
19             else
20                 QID_tuple ← dim;
21                 RDD ← QID_tuple, SA ;
22                 returnAnonymized (RDD,k);
23 RDD* ← Join(NA*_RDD) ;
24 return RDD*;
```

The $IRDD$ partition size is dependent on the $HDFS$ file and corresponding block sizes whereas processing of the $IRDD$ involves splitting the data into a several children

RDDs NA_RDD. To avoid extra load on memory, we cache the $IRDD$ to memory and disk. The combination of memory and disk provides the highest performance compared to other cache storage [116]. We future reduce the size of $IRDD$ after it applies generalization by re-partitioning the generalized results and caching the results (i.e., to avoid re-computation for the generalized results) each time. When there is no more generalization to be applied, all the results on cache are joined together to obtain RDD^*. The steps involved in Algorithm 15 is further described below in details.

1. First, all data records in $IRDD$ is checked for whether it violates the K-anonymity constraint.
2. In case, the K-anonymity has not been violated:
   - Check QID_{tuple} in the record if it satisfies K anonymity - that is the remaining QID does not require generalization and the record satisfies the anonymity requirement for the given K.
   - Store all records which satisfy the anonymity requirement in NA_RDD.
   - Apply a generalization level to the remaining records of A_RDD that does not satisfy the K-anonymity condition by choosing one dimension at a time from QID_{tuple}.
   - Apply generalization starting from the first dimension by calculating the equivalent class EC and compare the size of EC with K.
   - Replace the child C_{qid} with the parent P_{qid} for the highest number of a C_{qid} attributes appearing in the respective dimension.
   - Replace the QID_{tuple} with QID^*_{tuple} for a generalized dimension and store the results in A_RDD
3. Repeat the steps for A_RDD so that all the records are anonymized and transferred to NA_RDD.
4. Join NA_RDD as a result of each iteration for the final anonymization.

6.5.1.2 Example Demonstration

Input $IRDD$ in Table 6.2 results in two RDDs: a Non-Allowable Cut RDD1 and an Allowable Cut RDD1, respectively. The content of these two RDDs can be found in Table 6.3. To demonstrate the total number of RDDs being created, we tag them with numbers. We use Fig. 6.9 to generalize the records.

Allowable Cut RDD1 proceeds by creating two more RDDs: Non-Allowable RDD2 and Allowable Cut RDD 2. At this stage, we cache Non-Allowable Cut RDD1 and Non-Allowable Cut RDD2 in Table 6.4.

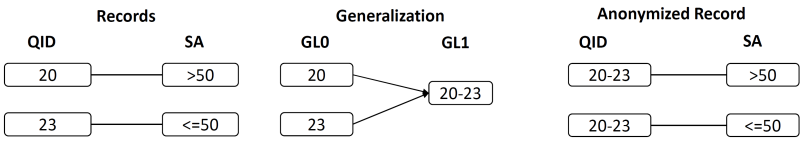


Figure 6.9: Generalization

Table 6.2: Input Data

Input RDD ($IRDD$)			
	QID1	**QID-2**	**SA**
$IRDD$ P1	21	male	>50
	23	female	<=50
$IRDD$ P2	24	female	>50
	26	male	<=50
$IRDD$ P3	27	female	>50
	32	male	>50
$IRDD$ P4	33	female	<=50
	21	male	<=50

Table 6.3: First Iteration

Non-Allowable Cut RDD 1 (NA_RDD1)			
	QID1	**QID-2**	**SA**
NA_RDD1 P1	21	male	>50
	21	male	<=50
Allowable Cut RDD 1 (A_RDD1)			
A_RDD1 P1	23-24	female	<=50
A_RDD1 P2	23-24	female	>50
	26-27	male	>50
A_RDD1 P3	32-33	male	>50
	26-27	female	<=50
A_RDD1 P4			
	32-33	female	<=50

The records within the Allowable Cut RDD2 are applied for generalization then further creates Non-Allowable Cut RDD3. There are no more Allowable Cut RDDs created from the Allowable Cut RDD2 since all the records in it satisfies the K anonymity requirement as shown in Table 6.5. As there are no more generalization to be applied, the final anonymized sets are joined together for all the cached Non-Allowable Cut RDDs which includes Non-Allowable Cut RDD1, Non-Allowable Cut RDD2, and Non-Allowable Cut RDD3 as shown in Table 6.6.

Table 6.4: Second Iteration

Non-Allowable Cut RDD 1 (NA_RDD1)			
	QID1	**QID-2**	**SA**
NA_RDD1 P1	21	male	>50
	21	male	<=50
Non-Allowable Cut RDD 2 (NA_RDD2)			
NA_RDD2 P1	23-24	female	<=50
	23-24	female	>50
Allowable Cut RDD 2 (A_RDD2)			
A_RDD2 P1	26-27	female	>50
	32-33	male	>50
A_RDD2 P2	26-27	male	<=50
	32-33	female	<=50

Table 6.5: Third Iteration

Non-Allowable Cut RDD 1 (NA_RDD1)			
	QID1	**QID-2**	**SA**
NA_RDD1 P1	21	male	>50
	21	male	<=50
Non-Allowable Cut RDD 2 (NA_RDD2)			
NA_RDD2 P1	23-24	female	<=50
	23-24	female	>50
Non-Allowable Cut RDD 3 (NA_RDD3)			
NA_RDD3 P1	26-27	gender	>50
	32-33	gender	>50
NA_RDD3 P2	26-27	gender	<=50
	32-33	gender	<=50

Table 6.6: Final Output

Join/ output RDD ($RDD*$)			
	QID1	**QID-2**	**SA**
$RDD*$ P1	21	male	>50
	21	male	<=50
$RDD*$ P2	23-24	female	<=50
	23-24	female	>50
$RDD*$ P3	26-27	gender	>50
	26-27	gender	<=50
$RDD*$ P4	32-33	gender	>50
	32-33	gender	<=50

The storage memory writes and manages Allowable Cut RDD1 partition A_RDD1(p1,

p2, and p3) and Non-Allowable Cut RDDs partitions NA_RDD1 (p1). To prevent the increase of data size for the storage memory, our approach caches Non-Allowable cut RDDs (NA_RDD1, NA_RDD2, and NA_RDD3) into the storage level "memory and disk". The storage memory holds the cache partitions as primary selection, if the storage memory requires eviction because of insufficient space, the cache partitions are moved to disk. We assume the size of executor memory is always limited. The increase in cached RDDs may result in LRU for each round of execution. To address this problem, we reduce the size of Allowable Cut RDD1 partitions (p1, p2, and p3) and Non-Allowable Cut RDDs partition (p1) from its parent $IRDD$ partitions (p1, p2, p3, and P4) using re-partitioning approach. We repeat this process at the end of each iteration to avoid LRU in the final stages of anonymization because the size of the NA_RDD1 is smaller compare to NA_RDD3. This is because the initial generalization level affects less records compare to the final generalization level.

We illustrate the effects of storage memory used by RDDs in terms of the size of the original input data size:

- **Worst case:** The size of a RDD is larger than the size of the executor. The executor cannot offer to cache the RDD in the executor memory as the size of the executor memory is not large enough to cache the RDD as well as to save any intermediate data produced while processing some tasks. In this case, the previously cached data (e.g., some parent RDDs saved in the earlier operation) is always removed from the executor memory to make more space for execution. This requires deleting of the cached previous RDD or trigger LRU operation to make more space in the executor memory and requires re-computation of the (deleted/removed) cached (previous) RDDs for the final stage.

- **Balance case:** The size of a RDD is same as the size of the executor (execution and storage memory combined). In this case, the executor borrows some space from the execution memory in case more space is required by the storage cache. The re-computation of partition may not necessary. The executor is limited not to use too many cache operations in memory but instead cache some RDDs in disk.

- **Best case:** The size of a RDD is smaller than the executor memory. This effectively eliminate the cache restriction in the memory and all RDDs are cached.

6.5.2 Mitigating Overheads

Our hybrid approach can effectively reduce many overheads we discussed in Section 6.4.3. The degree of how our proposed approach reduces the various overheads is summarised as shown in Table 6.7.

Our approach provides better control of data size and the number of partitions associated with RDDs which subsequently reduces many overheads appeared in other similar

Table 6.7: Increasing RDDS Overhead effect on our proposal vs existing approaches

	Re-computation Cost	**Shuffle Overhead**	**Message Exchange**	**Excessive Caching**
Recursive	High to Higher	High to Higher	High to Higher	High to Higher
Iteration	High to Medium	High to Medium	High to Medium	Medium
Proposed Solution	Medium to Low	Medium to Low	Medium	Medium to Low

methods. In our approach, children RDD size is always reduced by effectively removing the data from its parent RDD that is no longer required for computation. At the same time, the number of partitions at each child RDD is always reduced (i.e., almost half of the number of partitions used by its parent RDD). Our approach also balances the growth of RDDS. The number of RDDs created by our approach is almost half compare to when recursive operations were used alone. The number of RDDs in our approach is more than when iteration operations were used alone but the RDDs created by our approach operates on more optimal size of partitions that reduce overheads. Equation. 6.3 represents the number of growths of RDDs in our proposal.

$$OurProposal = (\sum_{j=0}^{i} j + \frac{n.j}{(k * GL)}) \tag{6.3}$$

where n represents the number of records, j represents the total number of split points, and i is used as a counter for each split. GL denotes a generalization level in K-group.

6.5.2.1 Reducing Re-computation Cost

To avoid re-computation of RDD partitions in each iteration, our approach reduces the size of Passive RDDs (NA_RDD) and process only Active RDDs(NA_RDD). The children RDDs of Active RDDs often require less memory space for computation as the size of children RDDs are always smaller than their parent RDDs. The reduction in data size also results in having to deal with smaller number of partitions. The reduced number of partitions can avoid the deletion and re-computation of Active RDD partitions from the executor memory, as a result, reduces the re-computation time.

6.5.2.2 Minimizing Shuffle Operations

The shuffle operations are mainly triggered by (1) the increasing number of RDD partitions, (2) multiple computations on relatively larger size of data, and (3) the number of stages of generalization process. Our approach addresses the first two cases (1) and (2) by reducing the number of partitions at each round of iteration. This is done by using re-partitioning approach in which assigns partitions according to data size while removing the partitions in which the processes have been done. In addition, we separate the processes relate to anonymized and non-anonymized records and processes only non-anonymized records in subsequent stages. These two strategies can minimize the shuffle overhead significantly. The last issue (3) is addressed as the result of the size of partition gets smaller in our hybrid approach as it can provide more space for subsequent iterations

for further execution. The shuffle cost of joining RDDs in the final stage is also reduced than recursive approach because the number of RDDs created by our hybrid approach is much smaller (i.e., about half).

6.5.2.3 Reducing Message Exchange

The reduction in the size and number of RDD can reduce the overheads associated with message exchange between the executor and the driver. The reduction in shuffle operations also reduces the communication cost associated with the message exchange because the smaller number of shuffle operations demand much less message exchange. The massages associated to change "status" (i.e., represented as M6 and M12 in Fig. 6.5) between the executor and the driver is reduced, so as the instruction massages (represented as M2, M8, and M13), as the result of the small number of partitions attached to Active RDDs in each iteration round.

6.5.2.4 Improve Cache Management

An important requirement for our proposed approach is to use cache for Passive RDDs that are only required in the final stage. We improve the cache usage by reducing the size of cache RDD at each stage. At the start of each iteration, we separate the anonymized records and store it NA_RDD while we re-partition according to the size of NA_RDD and cache it at the end of each iteration. The reduced number of partitions requires much less cache storage compare to the parent RDD. We use storage memory to hold the A_RDD for the next iteration. The storage memory keeps NA_RDD in the cache while the rest are saved in the disk.

6.6 Experiments and Results Analysis

In this section, we first describe our experimental setups that include details of the system environment configurations and dataset. This is followed by discussion of performance and scalability of our proposed approach. Finally, we discuss our findings in privacy and utility trade-offs.

Table 6.8: Adult dataset

QID	**Distinct** *qid* **Value**	*GL*
Age	74	6
Work Class	8	4
Education	16	4
Race	16	2
Gender	2	1
Native Country	41	2

Table 6.9: Setup Parameters

	Spark Parameters					Experiment Parameters		
	Driver Memory	Executor Memory	No of Executors	Executor Cores	No of Partitions	K Group Size	Dataset Records	Number of $QIDs$
Fig. 6.10	6.5 GB	4GB	12	3	8 - 14	2	0.5B - 2B	6
Fig. 6.11	6.5 GB	4GB	12	3	10	2	6B	2-6
Fig. 6.12, 6.13	6.5 GB	4GB	12	3	1	2 - 100	3.2K	6

6.6.1 Experimental environment and dataset

6.6.1.1 Datasets

We used US Census dataset (often described as Adult dataset) [15]. We downloaded the original Adult dataset, then scaled it up using [106] method to create a set of larger datasets for the experiments. Table 6.8 illustrates the quasi-identifiable attributes (QID) we used in our experiments and generalization level (GL) for each QID obtained from the taxonomy trees for Adult datasets. The" Salary" in the Adult dataset is set as sensitive attributes. The details of the experimental setup are illustrated in Table 6.9.

6.6.1.2 System Environment Configurations

We used Spark version 2.3 along with Yarn as a cluster manager. We configured Yarn and Hadoop Distributed File System (HDFS) using Apache Ambari. HDFS distributes data in a NameNode (worked as a master node), a secondary NameNode, and six DataNodes (worked as worker nodes). We configured 3GB memory for Yarn NodeManager while 1GB memory was allocated to ResourceManager, Driver, and Executor memories each.

6.6.2 Performance Analysis and Scalability

In this section, we discuss and compare the performance results of our proposal against similar implementation of iteration and recursion-based approaches. Our experiments were conducted to understand the performance implication based on the increasing data size and the scalability implication based on the operations conducted on the different number of QID. We ran all experiments at least 10 times and report the average results.

Increasing Data Size: we conducted the experiments to quantify the performance cost of our proposed approach. We first measure iteration-based Fig. 6.7 and recursion-based approach Fig. 6.6. Then compare them with our proposal Fig. 6.8. To ensure comparability across different experiments, we used the same workload and Spark cluster configuration. We also ensured that we use the same dataset size and the same number of QID for our experiments to compare the results.

Fig. 6.10 compares the execution times of our approach along with recursion and iteration-based approaches. The results indicate that our proposal yields the lowest execution time compared to other two while the recursion-based approach shows the highest execution time. The highest execution time in recursion-based approach appears due to

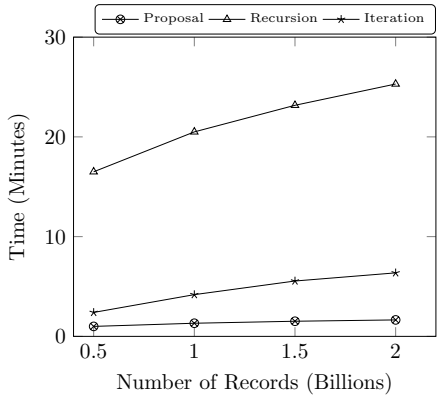


Figure 6.10: Compare results for Increasing record size

growth in the higher number of RDDs at each recursion. As it require to process all RDDs for the next recursion, it increases the execution time where RDDs were executed in a sequential manner. We also noticed that the execution time for iteration approach was relatively higher. We identified a significant bottleneck where relatively larger RDDs that are bigger than the size of the executor memory were processed.

Increase Number of QID : In the Fig.6.11 shows the set of experiments for varying the size of QIDs. We run an experiment for increasing the size of QID on a fixed number of K and the number of records to understand the relationship between the execution time and the scalability of operations represented by the size of QIDs. Note that processing of a QID involves applying generalization levels on the number of distinct attribute values with QID. Each additional QID required more computation to apply generalizations thus adds more execution time.

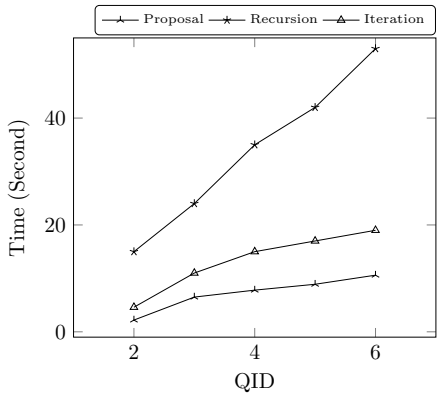


Figure 6.11: Compare results for Increasing QID for fix number of Records

The results indicate that our proposal yields the lowest execution time compare to other two approaches. We observe that the cardinality of quasi-identifiers and generalization level has a strong relationship with execution time. For example, the execution

time has increased between Q4 - Q6 in the iteration-based approach because the QIDs in this approach have more distinct number of quasi-identifiers compare to Q2 and Q3. Similarly, generalization levels applied for Q2 and Q3 were less compare to Q4 - Q6. We identify that "Native Country" in Q5 has a very high cardinality which resulted in the increase of execution time.

The increase in execution time for recursive approach was linear correspond to the increasing number of iterations of QID. We observe that there was a strong relationship between the depth of recursion (i.e., many how recursions were executed) and the execution time. For example, the QIDs of "Race" and "Native Country" have relatively high cardinal values with fewer levels of generalization which create more recursions. We observe that the execution time was higher in the inclusion of these QIDs compare to other QID sets.

The execution time for iterative approach was almost double compare to our approach. We assume that the reduction in the partitions size of each RDD has attributed to improve the execution time for our approach while the partitions size of RDDs in iterations remains the same. The execution time for Q4 - Q6 in iterative approach was more prominent compare to our approach. We identify that the iterative approach required checking dimensions from the start to the end in each and every iteration regardless whether the data was anonymized or not. This has significantly attributed towards increasing execution time.

6.6.3 Privacy and Utility benchmark

We utilized privacy and utility metrics to understand the implication of the level of privacy and utilise for our proposed method as these metrics allow us to quantify the success of an anonymization algorithm. A privacy level is estimated by recognizing the uniqueness of information, a low privacy normally implies that it is anything but difficult to distinguish an individual (e.g., attribute) from a group (e.g., numerous records). Similarly, a high privacy demonstrates that it is (increasingly) hard to uniquely recognize an individual from a group (e.g., there are numerous records having the same values). We used Kullback-Leibler-divergence (KLD) as a privacy metric while use Information Loss (I_L) as a utility benchmark.

6.6.3.1 Kullback-Leibler-Divergence

Kullback-Leibler-Divergence (KLD) is used to identify the likelihood "probability" of the original attribute present in the anonymized attribute for each record [76]. For example, let's assume that the original attribute of the age of 21 is anonymized into a range of 20-23. The KLD can measure what is the possibility of guessing the original data of age 21 from the range 20-23. The KLD is performed on the anonymized records by calculating the likelihood of the presence of each attribute then sums all the value for each attribute

within a record and repeats this for all records.

$$KLD = \sum_{r=1}^{n} P_{RDD^*}(r) \log \frac{P_{RDD^*}(r)}{P_{IRDD}(r)} \tag{6.4}$$

Here, P_{IRDD} indicates the sum of the likelihood of the presence of the original attribute within the original data (at a record level). P_{IRDD} at this stage would have a very high data utility and no privacy as there are no changes made. $P_{IRDD(r)}$ indicates the sum of the likelihood of the presence of the original attribute within the anonymized record. P_{RDD^*} usually has lost some degree of data utility and has gained some degree of privacy because the data in this set has changed from the baseline after an anonymization technique like our proposal is applied.

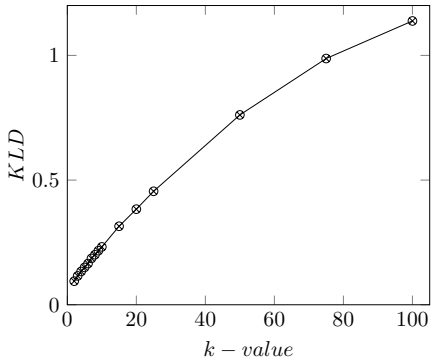


Figure 6.12: Performance of Privacy using Kullback-Leibler-Divergence score

The KLD score starts from the zero indicating two records values are same. As anonymization progresses, KLD score is expected to grow as the difference between the original dataset and the anonymized datasets becomes bigger. Fig.6.12 presents KLD scores of our proposal which demonstrates a sharp linear growth of the score at the lower level of K-group sizes (e.g., from $K = 2$ till $K = 50$) while the growth is slowed down after $K = 75$. A slight change of KLD scores in the range $K = 25$ to 100 represents the increase in generalization levels in order to obtain more similar records compare to lower K group size. As KLD measures the difference between records, the changes in the larger record set with possibly higher generalization levels means more records are different from the original record thus resulting increase in KLD score. The KLD score for our approach and big data Mondrian based approaches [55,113] provides similar scores to validates the accuracy of our implementation.

6.6.3.2 Information Loss:

Information Loss (I_L) is used to measure the degree of how uncertain it is to identify the original value from the anonymized value within a record [135]. The I_L in our context

is computed for each attribute individually and then added for all attributes for each generalization level. To compute I_L,

$$I_L(GL) = sum_{qid} - \log \frac{1}{(|(qid)ranginGL|)} \tag{6.5}$$

where GL represents the generalization level, qid represents either a numeric or categorical attribute. A generalization range for the numerical qid can be calculated by subtracting qid.min from qid.max in a given generalization level. For example, for a generalized data range of an age (17-20), the information loss is computed by calculating range.max minus range.min (i.e. 20 - 17= 3), taking the multiplicative inverse of the value (i.e.,1/3), taking the log of the inverted value (log(1/3) = log(0.333) = -4.77). As the result of log returns negating value, the formula converts it to the positive value [4.77=-(-4.77)]. For the categorical attribute, a generalization range is a complete list of all possible values for a given generalization hierarchy level. For example, for a generalization record a gender (Female or Male), the information loss is calculated by using the total number of unique categories (i.e., in this case 2), taking the multiplicative inverse of 2 (i.e., 1/2=0.5), taking the log of the inverted value (log(1/2) = log(0.5) = -0.301) then turn the result to the positive value [0.301=-(-0.301)]. The continuous and categorical value are then summed together to measure the total information loss.

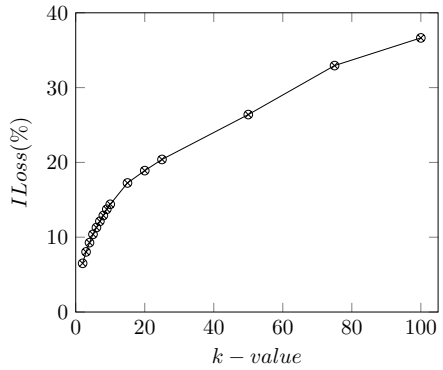


Figure 6.13: Performance of Utility using Information Loss score

The results reported in Fig. 6.13 shows the percentage of information loss from the original data to the anonymized data. The smaller K group sizes (e.g., from $K = 2$ till $K = 10$) appear to retain more information with the score range from 8% to 16 %. The score degrades quite sharply at $K > 10$. This appears due to the increase in the generalization level associated with K group size. The larger the K is, it contains the tuples with higher generalization levels which results more highly deviated record sets from the baseline. The I_L score depends on various aspects that include data type, generalization level, anonymization approaches. We compare the results of I_L score from our proposed approach with and identify the similar trend for K group size

The I_L score for our approach and big data Mondrian based existing studies [6,131,154] provides similar scores to validates the accuracy of our implementation for K group size.

6.7 Conclusion and Future work

We proposed a novel hybrid approach for multi-dimensional data anonymization strategy based on Mondrian. Our method creates a lot fewer numbers of RDDs and the smaller size partitions attached to each RDD. These effectively resolve many overheads associated with existing methods, including re-computation, shuffling, message exchange and cache management costs. Our experimental results illustrated that our proposal outperformed other similar methods in terms of performance and scalability. In addition, our proposal resulted in highly acceptable data privacy and utility scores. In future, we plan to investigate the characteristics of attributes (e.g., cardinality, generalization levels) to identify any relationship between the characteristics and the growth in the number of RDDs. We also intend to validate our proposal on other types of data anonymization technique other than Mondrian such as run-time-based generalization [83] or cell-based generalization [141].

Chapter 7

Conclusions and Future Directions

This thesis aimed to provide new privacy preserving models for protecting the privacy of data being processed in the major distributed processing platforms such as MapReduce and Spark.

In overall, the thesis is comprised of two sets of studies. The first set of studies, contained in Chapter 2 and Chapter 3, provide a set of new privacy preserving models for MapReduce while the second set of studies, consist of Chapter 4, Chapter 5, and Chapter 6, provides a set off novel privacy preservation solutions for Apache Spark platform.

In Chapter 2, we developed a privacy preserving platform for the prevention of privacy leakage in MapReduce. Our platform can be seen as an extra layer of data protection for MapReduce which allows the implementation of different anonymization techniques to be applied which produce different sets of anonymized datasets depending on the requirement of an application.

In Chapter 3, we introduced a novel k-NN classifier that can work on anonymized datasets. We demonstrated that our proposed k-NN classifier could work effectively on two separate anonymized datasets that were produced by utilising K-anonymity and differential privacy approaches. In addition, we provided that various privacy parameters (K group size, sensitivity and privacy budget) could be used to fine-tune the data quality for accurate classification results.

In Chapter 4, we proposed a data anonymization approach named SparkDA for Spark platform to provide more efficient in-memory data anonymization processing that could avoid expensive disk I/O. SparkDA uses two critical RDD transformations, FlatMapRDD and ReduceByKeyRDD, to take advantage of the Spark feature such as better partition control, in-memory processing, and cache management to overcome the frequent disk access during many data computation phases.

In Chapter 5, we developed a new generic approach for subtree-based data anonymization for Spark. Our generic approach addresses the limitations associated with the support for intensive iterative operations involved in the various subtree-based data anonymization techniques. Our proposal implements a series of RDDs, in which executed in parallel

fashion, to provide more efficient partition management, improved memory usage, and supports for better iteration management. The experimental results show improved performance by reducing the operation and computation complexity whiling maintaining competitive level of data privacy and high data utility scores.

In Chapter 6, We proposed a novel hybrid privacy-preservation solution for multi-dimensional data anonymization based on Mondrian that provides much better control for the creation of RDDs and the size of partitions associated with each RDD. Our hybrid approach reduces the overheads often found in the existing iterative based and recursive based approaches. We demonstrated that our proposal was effective in reducing overheads such as re-computation, shuffling, message exchange and cache management costs.

Our current study can be extended to the followings research directions:

- Implementation of other types of K-anonymity and differential privacy based approaches for single and multi-dimensional data in Spark using different RDD transformations.
- A generic privacy preserving framework for Spark that can implement different data anonymization techniques to produce different sets of anonymization results. A machine learning method could be investigated to find the optimal data anonymization strategy for different types of application.
- Designing a privacy preservation solution for run time generated data. By understanding the nature of the run-time generated data, it can find the most suitable data anonymization approach, apply it, and generate/store anonymized data.

Bibliography

[1] Spark overview. in: Overview - spark 2.1.0 documentation.

[2] Central statistics office (internet), 2011.

[3] Adnan, M., Afzal, M., Aslam, M., Jan, R., and Martinez-Enriquez, A. Minimizing big data problems using cloud computing based on hadoop architecture. In *2014 11th Annual High Capacity Optical Networks and Emerging/Enabling Technologies (Photonics for Energy)* (2014), IEEE, pp. 99–103.

[4] Agarwal, S., and Khanam, Z. Map reduce: A survey paper on recent expansion. *International Journal of Advanced Computer Science and Applications 6*, 8 (2015).

[5] Aggarwal, C. C., and Philip, S. Y. A general survey of privacy-preserving data mining models and algorithms. In *Privacy-preserving data mining*. Springer, 2008, pp. 11–52.

[6] Aghdam, M. R. S., and Sonehara, N. Achieving high data utility k-anonymization using similarity-based clustering model. *IEICE TRANSACTIONS on Information and Systems 99*, 8 (2016), 2069–2078.

[7] Agrawal, R., and Srikant, R. Privacy-preserving data mining. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data* (2000), pp. 439–450.

[8] Al-Zobbi, M., Shahrestani, S., and Ruan, C. Sensitivity-based anonymization of big data. In *2016 IEEE 41st Conference on Local Computer Networks Workshops (LCN Workshops)* (2016), IEEE, pp. 58–64.

[9] Al-Zobbi, M., Shahrestani, S., and Ruan, C. Improving mapreduce privacy by implementing multi-dimensional sensitivity-based anonymization. *Journal of Big Data 4*, 1 (2017), 45.

[10] Al-Zobbi Mohammed, Shahrestani Seyed, R. C. Experimenting sensitivity-based anonymization framework in apache spark. *Journal of Big Data 5*, 1 (October 2018), 38.

[11] Antonatos, S., Braghin, S., Holohan, N., Gkoufas, Y., and Mac Aonghusa, P. Prima: An end-to-end framework for privacy at scale. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)* (2018), IEEE, pp. 1531–1542.

[12] Ashkouti, F., khamforoosh, K., and Sheikhahmadi, A. Di-mondrian: Distributed improved mondrian for satisfaction of the l-diversity privacy model using apache spark. *Information Sciences 546* (2021), 1 – 24.

[13] Ashkouti, F., Sheikhahmadi, A., et al. Di-mondrian: Distributed improved mondrian for satisfaction of the l-diversity privacy model using apache spark. *Information Sciences 546* (2020), 1–24.

[14] Ashwin, M., Daniel, K., Johannes, G., and Muthuramakrishnan, V. l-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data 1*, 1 (2007), 1–52.

[15] Asuncion, A., and Newman, D. UCI machine learning repository, 2007.

[16] Ayala-Rivera, V., McDonagh, P., Cerqueus, T., Murphy, L., et al. A systematic comparison and evaluation of k-anonymization algorithms for practitioners. *Transactions on data privacy 7*, 3 (2014), 337–370.

[17] Baryalai, M., Jang-Jaccard, J., and Liu, D. Towards privacy-preserving classification in neural networks. In *2016 14th Annual Conference on Privacy, Security and Trust (PST)* (2016), IEEE, pp. 392–399.

[18] Bayardo, R. J., and Agrawal, R. Data privacy through optimal k-anonymization. In *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on* (2005), IEEE, pp. 217–228.

[19] Bazai, S. U., and Jang-Jaccard, J. Sparkda: Rdd-based high-performance data anonymization technique for spark platform. In *International Conference on Network and System Security* (2019), Springer, pp. 646–662.

[20] Bazai, S. U., and Jang-Jaccard, J. In-memory data anonymization using scalable and high performance rdd design. *Electronics 9*, 10 (2020), 1732.

[21] Bazai, S. U., Jang-Jaccard, J., and Alavizadeh, H. A generic approach for subtree-based data anonymization for apache spark. *IEEE Access* (2020).

[22] Bazai, S. U., Jang-Jaccard, J., and Alavizadeh, H. A novel hybrid approach for multi-dimensional data anonymization for apache spark. *Transactions on Privacy and Security (TOPS)* (2020).

[23] Bazai, S. U., Jang-Jaccard, J., and Wang, R. Anonymizing k-nn classification on mapreduce. In *International Conference on Mobile Networks and Management* (2017), Springer, pp. 364–377.

[24] Bazai, S. U., Jang-Jaccard, J., and Zhang, X. A privacy preserving platform for mapreduce. In *International Conference on Applications and Techniques in Information Security* (2017), Springer, pp. 88–99.

[25] Benlachmi, Y., and Hasnaoui, M. L. Big data and spark: Comparison with hadoop. In *2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)* (2020), IEEE, pp. 811–817.

[26] Boubela, R. N., Kalcher, K., Huf, W., Našel, C., and Moser, E. Big data approaches for the analysis of large-scale fmri data using apache spark and gpu processing: a demonstration on resting-state fmri data from the human connectome project. *Frontiers in neuroscience 9* (2016), 492.

[27] Cadwalladr, C., and Graham-Harrison, E. Revealed: 50 million facebook profiles harvested for cambridge analytica in major data breach. *The guardian 17* (2018), 22.

[28] Chakravorty, A., Rong, C., Jayaram, K., and Tao, S. Scalable, efficient anonymization with incognito-framework & algorithm. In *2017 IEEE International Congress on Big Data (BigData Congress)* (2017), IEEE, pp. 39–48.

[29] Chakravorty, A., Wlodarczyk, T. W., and Rong, C. A scalable k-anonymization solution for preserving privacy in an aging-in-place welfare intercloud. In *2014 IEEE International Conference on Cloud Engineering* (2014), IEEE, pp. 424–431.

[30] Chefranov, A., Abhari, S. M. A., Alavizadeh, H., and Zanjani, M. F. Secure true random number generator in wlan/lan. In *Proceedings of the 6th International Conference on Security of Information and Networks* (2013), pp. 331–335.

[31] Chen, C., Li, K., Ouyang, A., Zeng, Z., and Li, K. Gflink: An in-memory computing architecture on heterogeneous cpu-gpu clusters for big data. *IEEE Transactions on Parallel and Distributed Systems 29*, 6 (2018), 1275–1288.

[32] Chen, G., Cai, Q., and Zhan, Y. Approaches on personal data privacy preserving in cloud: a survey. In *Proceedings of The Third International Conference on Data Mining, Internet Computing, and Big Data, Konya* (2016), pp. 36–43.

[33] Chen, K., and Liu, L. Privacy preserving data classification with rotation perturbation. In *Fifth IEEE International Conference on Data Mining (ICDM'05)* (2005), IEEE, pp. 4–pp.

[34] Cheng, C., Xiaoli, L., Linfeng, W., Longxin, L., and Xiaofeng, W. Algorithm for k-anonymity based on ball-tree and projection area density partition. In *2019 14th International Conference on Computer Science & Education (ICCSE)* (2019), IEEE, pp. 972–975.

[35] Choi, I. S., Yang, W., and Kee, Y.-S. Early experience with optimizing i/o performance using high-performance ssds for in-memory cluster computing. In *2015 IEEE International Conference on Big Data (Big Data)* (2015), IEEE, pp. 1073–1083.

[36] Choi, W., Hong, S., and Jeong, W.-K. Vispark: Gpu-accelerated distributed visual computing using spark. *SIAM Journal on Scientific Computing 38*, 5 (2016), S700–S719.

[37] Clifton, C., and Tassa, T. On syntactic anonymity and differential privacy. In *Data Engineering Workshops (ICDEW), 2013 IEEE 29th International Conference on* (2013), IEEE, pp. 88–93.

[38] Cover, T., and Hart, P. Nearest neighbor pattern classification. *IEEE transactions on information theory 13*, 1 (1967), 21–27.

[39] Cuzzocrea, Alfredo ad Saccà, D. Balancing accuracy and privacy of olap aggregations on data cubes. In *Proceedings of the ACM 13th international workshop on Data warehousing and OLAP* (2010), pp. 93–98.

[40] Dean, J., and Ghemawat, S. Mapreduce: simplified data processing on large clusters. *Communications of the ACM 51*, 1 (2008), 107–113.

[41] Derbeko, P., Dolev, S., Gudes, E., and Sharma, S. Security and privacy aspects in mapreduce on clouds: A survey. *Computer Science Review 20* (2016), 1–28.

[42] Douriez, M., Doraiswamy, H., Freire, J., and Silva, C. T. Anonymizing nyc taxi data: Does it matter? In *2016 IEEE international conference on data science and advanced analytics (DSAA)* (2016), IEEE, pp. 140–148.

[43] Dwork, C. Differential privacy: A survey of results. In *International Conference on Theory and Applications of Models of Computation* (2008), Springer, pp. 1–19.

[44] Dwork, C. The differential privacy frontier. In *Theory of Cryptography Conference* (2009), Springer, pp. 496–502.

[45] Fletcher, S., and Islam, M. Z. Differentially private random decision forests using smooth sensitivity. *Expert Systems with Applications 78* (2017), 16–31.

[46] Fung, B. C., Wang, K., Chen, R., and Yu, P. S. Privacy-preserving data publishing: A survey of recent developments. *ACM Computing Surveys (Csur) 42*, 4 (2010), 1–53.

[47] Fung, B. C., Wang, K., and Philip, S. Y. Anonymizing classification data for privacy preservation. *IEEE transactions on knowledge and data engineering 19*, 5 (2007), 711–725.

[48] Fung, B. C., Wang, K., Wang, L., and Debbabi, M. A framework for privacy-preserving cluster analysis. In *Intelligence and Security Informatics, 2008. ISI 2008. IEEE International Conference on* (2008), IEEE, pp. 46–51.

[49] Fung, B. C., Wang, K., Wang, L., and Hung, P. C. Privacy-preserving data publishing for cluster analysis. *Data Knowledge Engineering 68*, 6 (2009), 552 – 575.

[50] Fung, B. C., Wang, K., and Yu, P. S. Top-down specialization for information and privacy preservation. In *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on* (2005), IEEE, pp. 205–216.

[51] Ganta, S. R., Kasiviswanathan, S. P., and Smith, A. Composition attacks and auxiliary information in data privacy. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining* (2008), pp. 265–273.

[52] Gao, Z.-Q., and Zhang, L.-J. Dphkms: An efficient hybrid clustering preserving differential privacy in spark. In *International Conference on Emerging Internetworking, Data & Web Technologies* (2017), Springer, pp. 367–377.

[53] Gentry, C. Computing arbitrary functions of encrypted data. *Communications of the ACM 53*, 3 (2010), 97–105.

[54] Ghazi, M. R., and Gangodkar, D. Hadoop, mapreduce and hdfs: a developers perspective. *Procedia Computer Science 48*, C (2015), 45–50.

[55] Ghinita, G., Karras, P., Kalnis, P., and Mamoulis, N. A framework for efficient data anonymization under privacy and accuracy constraints. *ACM Transactions on Database Systems (TODS) 34*, 2 (2009), 1–47.

[56] Gopalani, S., and Arora, R. Comparing apache spark and mapreduce with performance analysis using k-means. *International journal of computer applications 113*, 1 (2015).

[57] Gounaris, A., and Torres, J. A methodology for spark parameter tuning. *Big data research 11* (2018), 22–32.

[58] Grolinger, K., Hayes, M., Higashino, W. A., L'Heureux, A., Allison, D. S., and Capretz, M. A. Challenges for mapreduce in big data. In *2014 IEEE world congress on services* (2014), IEEE, pp. 182–189.

[59] Gu, J., Watanabe, Y. H., Mazza, W. A., Shkapsky, A., Yang, M., Ding, L., and Zaniolo, C. Rasql: Greater power and performance for big data analytics with recursive-aggregate-sql on spark. In *Proceedings of the 2019 International Conference on Management of Data* (2019), pp. 467–484.

[60] Gufler, B., Augsten, N., Reiser, A., and Kemper, A. Handling data skew in mapreduce. *Closer 11* (2011), 574–583.

[61] Hall, R., et al. *New Statistical Applications for Differential Privacy.* PhD thesis, PhD thesis, PhD thesis, Carnegie Mellon, 2012.

[62] He, X., Machanavajjhala, A., and Ding, B. Blowfish privacy: Tuning privacy-utility trade-offs using policies. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data* (2014), pp. 1447–1458.

[63] Heifetz, A., Mugunthan, V., and Kagal, L. Shade: A differentially-private wrapper for enterprise big data. In *2017 IEEE International Conference on Big Data (Big Data)* (2017), IEEE, pp. 1033–1042.

[64] Hong, S., Choi, W., and Jeong, W.-K. Gpu in-memory processing using spark for iterative computation. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)* (2017), IEEE, pp. 31–41.

[65] Huang, W., Meng, L., Zhang, D., and Zhang, W. In-memory parallel processing of massive remotely sensed data using an apache spark on hadoop yarn model. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing 10*, 1 (2016), 3–19.

[66] Inan, A., Kantarcioglu, M., and Bertino, E. Using anonymized data for classification. In *2009 IEEE 25th International Conference on Data Engineering* (2009), IEEE, pp. 429–440.

[67] Inan, A., Kantarcioglu, M., Ghinita, G., and Bertino, E. Private record matching using differential privacy. In *Proceedings of the 13th International Conference on Extending Database Technology* (2010), ACM, pp. 123–134.

[68] Jain, P., Gyanchandani, M., and Khare, N. Big data privacy: a technological perspective and review. *Journal of Big Data 3*, 1 (2016), 25.

[69] Jebali, A., Sassi, S., and Jemai, A. Secure data outsourcing in presence of the inference problem: issues and directions. *Journal of Information and Telecommunication* (2020), 1–19.

[70] Jiang, X., Ji, Z., Wang, S., Mohammed, N., Cheng, S., and Ohno-Machado, L. Differential-private data publishing through component analysis. *Transactions on data privacy 6*, 1 (2013), 19.

[71] Kalavri, V., and Vlassov, V. Mapreduce: Limitations, optimizations and open issues. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on* (2013), IEEE, pp. 1031–1038.

[72] Kang, M., and Lee, J.-G. An experimental analysis of limitations of mapreduce for iterative algorithms on spark. *Cluster Computing 20*, 4 (2017), 3593–3604.

[73] Kao, C.-H., Hsieh, C.-H., Chu, Y.-F., Kuang, Y.-T., and Yang, C.-K. Using data visualization technique to detect sensitive information re-identification problem of real open dataset. *Journal of Systems Architecture 80* (2017), 85–91.

[74] Katsogridakis, P., Papagiannaki, S., and Pratikakis, P. Execution of recursive queries in apache spark. In *European Conference on Parallel Processing* (2017), Springer, pp. 289–302.

[75] Kelly, D. J., Raines, R. A., Grimaila, M. R., Baldwin, R. O., and Mullins, B. E. A survey of state-of-the-art in anonymity metrics. In *Proceedings of the 1st ACM workshop on Network data anonymization* (2008), ACM, pp. 31–40.

[76] Kifer, D., and Gehrke, J. Injecting utility into anonymized datasets. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data* (2006), ACM, pp. 217–228.

[77] Kisilevich, S., Rokach, L., Elovici, Y., and Shapira, B. Efficient multidimensional suppression for k-anonymity. *IEEE Transactions on Knowledge and Data Engineering 22*, 3 (2009), 334–347.

[78] Ko, S. Y., Jeon, K., and Morales, R. The hybrex model for confidentiality and privacy in cloud computing. *HotCloud 11* (2011), 8–8.

[79] Lee, J. Y., Brown, J. J., and Ryan, L. M. Sufficiency revisited: Rethinking statistical algorithms in the big data era. *The American Statistician 71*, 3 (2017), 202–208.

[80] Lee, S., Kang, S., Kim, J., and Yu, E. J. Scalable distributed data cube computation for large-scale multidimensional data analysis on a spark cluster. *Cluster Computing 22*, 1 (2019), 2063–2087.

[81] LeFevre, K., DeWitt, D. J., and Ramakrishnan, R. Mondrian multidimensional k-anonymity. In *22nd International conference on data engineering (ICDE'06)* (2006), IEEE, pp. 25–25.

[82] LeFevre, K., DeWitt, D. J., and Ramakrishnan, R. Workload-aware anonymization. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining* (2006), ACM, pp. 277–286.

[83] Li, J., Wong, R. C.-W., Fu, A. W.-C., and Pei, J. Anonymization by local recoding in data with attribute hierarchical taxonomies. *IEEE Transactions on Knowledge and Data Engineering 20*, 9 (2008), 1181–1194.

[84] Li, M., Tan, J., Wang, Y., Zhang, L., and Salapura, V. Sparkbench: a comprehensive benchmarking suite for in memory data analytic platform spark. In *Proceedings of the 12th ACM international conference on computing frontiers* (2015), pp. 1–8.

[85] Li, M., Tan, J., Wang, Y., Zhang, L., and Salapura, V. Sparkbench: a spark benchmarking suite characterizing large-scale in-memory data analytics. *Cluster Computing 20*, 3 (2017), 2575–2589.

[86] Li, N., Li, T., and Venkatasubramanian, S. t-closeness: Privacy beyond k-anonymity and l-diversity. In *2007 IEEE 23rd International Conference on Data Engineering* (2007), IEEE, pp. 106–115.

[87] Liang, F., Feng, C., Lu, X., and Xu, Z. Performance benefits of datampi: a case study with bigdatabench. In *Workshop on Big Data Benchmarks, Performance Optimization, and Emerging Hardware* (2014), Springer, pp. 111–123.

[88] Machanavajjhala, A., Gehrke, J., Kifer, D., and Venkitasubramaniam, M. l-diversity: Privacy beyond k-anonymity. In *22nd International Conference on Data Engineering (ICDE'06)* (2006), IEEE, pp. 24–24.

[89] Machanavajjhala, A., Kifer, D., Gehrke, J., and Venkitasubramaniam, M. l-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data (TKDD) 1*, 1 (2007), 3–es.

[90] Maillo, J., Ramírez, S., Triguero, I., and Herrera, F. knn-is: An iterative spark-based design of the k-nearest neighbors classifier for big data. *Knowledge-Based Systems 117* (2017), 3–15.

[91] Maillo, J., Triguero, I., and Herrera, F. A mapreduce-based k-nearest neighbor approach for big data classification. In *2015 IEEE Trustcom/BigDataSE/ISPA* (2015), vol. 2, IEEE, pp. 167–172.

[92] Manovich, L. Digital traces in context— 100 billion data rows per second: Media analytics in the early 21st century. *International journal of communication 12* (2018), 16.

[93] Mavridis, I., and Karatza, H. Performance evaluation of cloud-based log file analysis with apache hadoop and apache spark. *Journal of Systems and Software 125* (2017), 133–151.

[94] Mehta, B. B., and Rao, U. P. Improved l-diversity: Scalable anonymization approach for privacy preserving big data publishing. *Journal of King Saud University-Computer and Information Sciences* (2019).

[95] Miryala, G., and Ludwig, S. A. Comparing spark with mapreduce: glowworm swarm optimization applied to multimodal functions. *International Journal of Swarm Intelligence Research (IJSIR) 9*, 3 (2018), 1–22.

[96] Nergiz, M. E., and Clifton, C. δ-presence without complete world knowledge. *IEEE Transactions on Knowledge and Data Engineering 22*, 6 (2009), 868–883.

[97] Nezarat, A., and Yavari, K. A distributed method based on mondrian algorithm for big data anonymization. In *International Congress on High-Performance Computing and Big Data Analysis* (2019), Springer, pp. 84–97.

[98] Nicolae, B., Costa, C. H., Misale, C., Katrinis, K., and Park, Y. Leveraging adaptive i/o to optimize collective data shuffling patterns for big data analytics. *IEEE Transactions on Parallel and Distributed Systems 28*, 6 (2016), 1663–1674.

[99] Niu, B., Li, Q., Zhu, X., Cao, G., and Li, H. Achieving k-anonymity in privacy-aware location-based services. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications* (2014), IEEE, pp. 754–762.

[100] Ohrimenko, O., Costa, M., Fournet, C., Gkantsidis, C., Kohlweiss, M., and Sharma, D. Observing and preventing leakage in mapreduce. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (2015), pp. 1570–1581.

[101] Onik, M. M. H., Kim, C.-S., Lee, N.-Y., and Yang, J. Privacy-aware blockchain for personal data sharing and tracking. *Open Computer Science 9*, 1 (2019), 80–91.

[102] Patel, A. B., Birla, M., and Nair, U. Addressing big data problem using hadoop and map reduce. In *Engineering (NUiCONE), 2012 Nirma University International Conference on* (2012), IEEE, pp. 1–5.

[103] Peethambaran, G., Naikodi, C., and Suresh, L. An ensemble learning approach for privacy–quality–efficiency trade-off in data analytics. In *2020 International Conference on Smart Electronics and Communication (ICOSEC)* (2020), IEEE, pp. 228–235.

[104] Perez, T. B., Chen, W., Ji, R., Liu, L., and Zhou, X. Pets: Bottleneck-aware spark tuning with parameter ensembles. In *2018 27th International Conference on Computer Communication and Networks (ICCCN)* (2018), IEEE, pp. 1–9.

[105] Pervaiz, Z., Aref, W. G., Ghafoor, A., and Prabhu, N. Accuracy-constrained privacy-preserving access control mechanism for relational data. *IEEE Transactions on Knowledge and Data Engineering 26*, 4 (2013), 795–807.

[106] Philip, M. Top down specialization on spark, 2019.

[107] Pomares-Quimbaya, A., Sierra-Múnera, A., Mendoza-Mendoza, J., Malaver-Moreno, J., Carvajal, H., and Moncayo, V. Anonylitics: From a small data to a big data anonymization system for analytical projects. In *presentado en 21st International Conference on Enterprise Information Systems* (2019), pp. 61–71.

[108] Preuveneers, D., and Joosen, W. Security and privacy controls for streaming data in extended intelligent environments. *Journal of Ambient Intelligence and Smart Environments 8*, 4 (2016), 467–483.

[109] Rogala, M., Hidders, J., and Sroka, J. Datalogra: datalog with recursive aggregation in the spark rdd model. In *Proceedings of the Fourth International Workshop on Graph Data Management Experiences and Systems* (2016), pp. 1–6.

[110] Roy, I., Setty, S. T., Kilzer, A., Shmatikov, V., and Witchel, E. Airavat: Security and privacy for mapreduce. In *NSDI* (2010), vol. 10, pp. 297–312.

[111] Samadi, Y., Zbakh, M., and Tadonki, C. Performance comparison between hadoop and spark frameworks using hibench benchmarks. *Concurrency and Computation: Practice and Experience 30*, 12 (2018), e4367.

[112] Sarwate, A. D., and Chaudhuri, K. Signal processing and machine learning with differential privacy: Algorithms and challenges for continuous data. *IEEE signal processing magazine 30*, 5 (2013), 86–94.

[113] Sattar, A. S., Li, J., Ding, X., Liu, J., and Vincent, M. A general framework for privacy preserving data publishing. *Knowledge-Based Systems 54* (2013), 276–287.

[114] Schatz, M. C. Cloudburst: highly sensitive read mapping with mapreduce. *Bioinformatics 25*, 11 (2009), 1363–1369.

[115] Selvi, U., and Pushpa, S. Big data feature selection to achieve anonymization. In *International Conference on Communication, Computing and Electronics Systems* (2020), Springer, pp. 59–67.

[116] Shi, J., Qiu, Y., Minhas, U. F., Jiao, L., Wang, C., Reinwald, B., and Özcan, F. Clash of the titans: Mapreduce vs. spark for large scale data analytics. *Proceedings of the VLDB Endowment 8*, 13 (2015), 2110–2121.

[117] Shkapsky, A., Yang, M., Interlandi, M., Chiu, H., Condie, T., and Zaniolo, C. Big data analytics with datalog queries on spark. In *Proceedings of the 2016 International Conference on Management of Data* (2016), pp. 1135–1149.

[118] Singh, D., and Reddy, C. K. A survey on platforms for big data analytics. *Journal of big data 2*, 1 (2015), 8.

[119] Sopaoglu, U., and Abul, O. A top-down k-anonymization implementation for apache spark. In *2017 IEEE International Conference on Big Data (Big Data)* (2017), IEEE, pp. 4513–4521.

[120] Sopaoglu, U., and Abul, O. A utility based approach for data stream anonymization. *Journal of Intelligent Information Systems* (2019), 1–27.

[121] Soria-Comas, J., Domingo-Ferrer, J., Sánchez, D., and Martínez, S. Enhancing data utility in differential privacy via microaggregation-based k-anonymity. *The VLDB Journal 23*, 5 (2014), 771–794.

[122] Stupar, A., Michel, S., and Schenkel, R. Rankreduce–processing k-nearest neighbor queries on top of mapreduce. *Large-Scale Distributed Systems for Information Retrieval 15* (2010).

[123] Sun, X., Wang, H., Li, J., and Truta, T. M. Enhanced p-sensitive k-anonymity models for privacy preserving data publishing. *Transactions on Data Privacy 1*, 2 (2008), 53–66.

[124] Suneetha, V., Suresh, S., and Jhananie, V. A novel framework using apache spark for privacy preservation of healthcare big data. In *2020 2nd International Conference on Innovative Mechanisms for Industry Applications (ICIMIA)* (2020), IEEE, pp. 743–749.

[125] Sweeney, L. Datafly: A system for providing anonymity in medical data. In *Database Security XI*. Springer, 1998, pp. 356–381.

[126] Sweeney, L. Achieving k-anonymity privacy protection using generalization and suppression. *International Journal of Uncertain Fuzziness and Knowledge ased Systems 10*, 05 (2002), 571–588.

[127] Sweeney, L. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 10*, 05 (2002), 557–570.

[128] Tan, X., Di, L., Zhong, Y., Yao, Y., Sun, Z., and Ali, Y. Spark-based adaptive mapreduce data processing method for remote sensing imagery. *International Journal of Remote Sensing 42*, 1 (2020), 171–187.

[129] Tao, Y., Lin, W., and Xiao, X. Minimal mapreduce algorithms. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data* (2013), pp. 529–540.

[130] Teijeiro, D., Pardo, X. C., Penas, D. R., González, P., Banga, J. R., and Doallo, R. Evaluation of parallel differential evolution implementations on mapreduce and spark. In *European Conference on Parallel Processing* (2016), Springer, pp. 397–408.

[131] Terrovitis, M., Mamoulis, N., and Kalnis, P. Privacy-preserving anonymization of set-valued data. *Proceedings of the VLDB Endowment 1*, 1 (2008), 115–125.

[132] To, Q.-C., Nguyen, B., and Pucheral, P. Trustedmr: A trusted mapreduce system based on tamper resistance hardware. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"* (2015), Springer, pp. 38–56.

[133] Tran, Q., and Sato, H. A solution for privacy protection in mapreduce. In *Computer Software and Applications Conference (COMPSAC), 2012 IEEE 36th Annual* (2012), IEEE, pp. 515–520.

[134] Wai, E. N. C., Tsai, P.-W., and Pan, J.-S. Hierarchical pso clustering on mapreduce for scalable privacy preservation in big data. In *International Conference on Genetic and Evolutionary Computing* (2016), Springer, pp. 36–44.

[135] Wan, Z., Vorobeychik, Y., Xia, W., Clayton, E. W., Kantarcioglu, M., Ganta, R., Heatherly, R., and Malin, B. A. A game theoretic framework for analyzing re-identification risk. *PloS one 10*, 3 (2015), e0120592.

[136] Wang, K., Yu, P. S., and Chakraborty, S. Bottom-up generalization: A data mining solution to privacy protection. In *Data Mining, 2004. ICDM'04. Fourth IEEE International Conference on* (2004), IEEE, pp. 249–256.

[137] Wang, S., Sinnott, R., and Nepal, S. Protecting the location privacy of mobile social media users. In *2016 IEEE International Conference on Big Data (Big Data)* (2016), IEEE, pp. 1143–1150.

[138] Xia, D., Li, H., Wang, B., Li, Y., and Zhang, Z. A mapreduce-based nearest neighbor approach for big-data-driven traffic flow prediction. *IEEE access 4* (2016), 2920–2934.

[139] Xiao, X., and Tao, Y. Dynamic anonymization: accurate statistical analysis with privacy preservation. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data* (2008), pp. 107–120.

[140] Xiao, Z., Xu, J., and Meng, X. p-sensitivity: A semantic privacy-protection model for location-based services. In *2008 Ninth International Conference on Mobile Data Management Workshops, MDMW* (2008), IEEE, pp. 47–54.

[141] Xu, J., Wang, W., Pei, J., Wang, X., Shi, B., and Fu, A. W.-C. Utility-based anonymization using local recoding. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining* (2006), pp. 785–790.

[142] Zaharia, M. *An architecture for fast and general data processing on large clusters.* Morgan and Claypool, 2016.

[143] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., and Stoica, I. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation* (2012), USENIX Association, pp. 2–2.

[144] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., Stoica, I., et al. Spark: Cluster computing with working sets. *HotCloud 10*, 10-10 (2010), 95.

[145] Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M. J., et al. Apache spark: a unified engine for big data processing. *Communications of the ACM 59*, 11 (2016), 56–65.

[146] Zhang, C., Li, F., and Jestes, J. Efficient parallel knn joins for large data in mapreduce. In *Proceedings of the 15th international conference on extending database technology* (2012), pp. 38–49.

[147] Zhang, K., Tanimura, Y., Nakada, H., and Ogawa, H. Understanding and improving disk-based intermediate data caching in spark. In *Big Data (Big Data), 2017 IEEE International Conference on* (2017), IEEE, pp. 2508–2517.

[148] Zhang, K., Zhou, X., Chen, Y., Wang, X., and Ruan, Y. Sedic: privacy-aware data intensive computing on hybrid clouds. In *Proceedings of the 18th ACM conference on Computer and communications security* (2011), ACM, pp. 515–526.

[149] Zhang, X., Dou, W., Pei, J., Nepal, S., Yang, C., Liu, C., and Chen, J. Proximity-aware local-recoding anonymization with mapreduce for scalable big data privacy preservation in cloud. *IEEE transactions on computers 64*, 8 (2014), 2293–2307.

[150] Zhang, X., Liu, C., Nepal, S., Dou, W., and Chen, J. Privacy-preserving layer over mapreduce on cloud. In *Cloud and Green Computing (CGC), 2012 Second International Conference on* (2012), IEEE, pp. 304–310.

[151] Zhang, X., Liu, C., Nepal, S., Yang, C., Dou, W., and Chen, J. Combining top-down and bottom-up: scalable sub-tree anonymization over big data using mapreduce on cloud. In *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications* (2013), IEEE, pp. 501–508.

[152] Zhang, X., Liu, C., Nepal, S., Yang, C., Dou, W., and Chen, J. Sac-frapp: a scalable and cost-effective framework for privacy preservation over big data on cloud. *Concurrency and Computation: Practice and Experience 25*, 18 (2013), 2561–2576.

[153] Zhang, X., Nepal, S., Yang, C., Dou, W., and Chen, J. A hybrid approach for scalable sub-tree anonymization over big data using mapreduce on cloud. *Journal of Computer and System Sciences 80*, 5 (2014), 1008–1020.

[154] Zhang, X., Qi, L., He, Q., and Dou, W. Scalable iterative implementation of mondrian for big data multidimensional anonymisation. In *International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage* (2016), Springer, pp. 311–320.

[155] Zhang, X., Yang, C., Nepal, S., Liu, C., Dou, W., and Chen, J. A mapreduce based approach of scalable multidimensional anonymization for big data privacy preservation on cloud. In *2013 International conference on cloud and green computing* (2013), IEEE, pp. 105–112.

[156] Zhang, X., Yang, L. T., Liu, C., and Chen, J. A scalable two-phase top-down specialization approach for data anonymization using mapreduce on cloud. *IEEE Transactions on Parallel and Distributed Systems 25*, 2 (2014), 363–373.

[157] Zhang, Y., Gao, Q., Gao, L., and Wang, C. imapreduce: A distributed computing framework for iterative computation. *Journal of Grid Computing 10*, 1 (2012), 47–68.

[158] Zhou, F., Han, Z., Zhao, J., and Wang, W. A spark-based approach for high-efficiency embedded feature selection. In *2019 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)* (2019), IEEE, pp. 36–43.

[159] Zhou, L., Wang, H., and Wang, W. Parallel implementation of classification algorithms based on cloud computing environment. *TELKOMNIKA Indonesian Journal of Electrical Engineering 10*, 5 (2012), 1087–1092.

[160] Zou, Y., Mhaidli, A. H., McCall, A., and Schaub, F. " i've got nothing to lose": Consumers' risk perceptions and protective actions after the equifax data breach. In *Fourteenth Symposium on Usable Privacy and Security ({SOUPS} 2018)* (2018), pp. 197–216.

MASSEY UNIVERSITY
GRADUATE RESEARCH SCHOOL

STATEMENT OF CONTRIBUTION
DOCTORATE WITH PUBLICATIONS/MANUSCRIPTS

We, the candidate and the candidate's Primary Supervisor, certify that all co-authors have consented to their work being included in the thesis and they have accepted the candidate's contribution as indicated below in the *Statement of Originality*.

Name of candidate:	Sibghat Ullah
Name/title of Primary Supervisor:	Associate Professor Julian Jang-Jaccard
Name of Research Output and full reference:	
Bazai, Sibghat Ullah, Julian Jang-Jaccard, and Xuyun Zhang. "A privacy preserving platform for mapreduce." In International Conference on Applications and Techniques in Information Security, pp. 88-99. Springer, Singapore, 2017.	
In which Chapter is the Manuscript /Published work:	Chapter 2
Please indicate:	
• The percentage of the manuscript/Published Work that was contributed by the candidate:	70
and	
• Describe the contribution that the candidate has made to the Manuscript/Published Work:	
Conceptualization, methodology, validation, formal analysis, investigation, original draft preparation, visualization, and responding to reviewers' comments	
For manuscripts intended for publication please indicate target journal:	
N/A	
Candidate's Signature:	Sibghat Ullah Bazai Digitally signed by Sibghat Ullah Bazai Date: 2020.09.03 18:31:08 +12'00'
Date:	03/09/2020
Primary Supervisor's Signature:	Julian Jang-Jaccard Digitally signed by Julian Jang-Jaccard Date: 2020.09.05 15:40:11 +12'00'
Date:	05/09/2020

(This form should appear at the end of each thesis chapter/section/appendix submitted as a manuscript/ publication or collected as an appendix at the end of the thesis)

MASSEY UNIVERSITY
GRADUATE RESEARCH SCHOOL

STATEMENT OF CONTRIBUTION
DOCTORATE WITH PUBLICATIONS/MANUSCRIPTS

We, the candidate and the candidate's Primary Supervisor, certify that all co-authors have consented to their work being included in the thesis and they have accepted the candidate's contribution as indicated below in the *Statement of Originality*.

Name of candidate:	Sibghat Ullah
Name/title of Primary Supervisor:	Associate Professor Julian Jang-Jaccard
Name of Research Output and full reference:	
Bazai, Sibghat Ullah, Julian Jang-Jaccard, and Ruili Wang. "Anonymizing k-NN classification on MapReduce." In International Conference on Mobile Networks and Management, pp. 364-377. Springer, Cham, 2017.	
In which Chapter is the Manuscript /Published work:	Chapter 3
Please indicate:	
• The percentage of the manuscript/Published Work that was contributed by the candidate:	70
and	
• Describe the contribution that the candidate has made to the Manuscript/Published Work:	
Conceptualization, methodology, validation, formal analysis, investigation, original draft preparation, visualization, and responding to reviewers' comments	
For manuscripts intended for publication please indicate target journal:	
N/A	
Candidate's Signature:	Sibghat Ullah Bazai Digitally signed by Sibghat Ullah Bazai Date: 2020.09.03 18:31:29 +12'00'
Date:	03/09/2020
Primary Supervisor's Signature:	Julian Jang-Jaccard Digitally signed by Julian Jang-Jaccard Date: 2020.09.04 14:04:36 +12'00'
Date:	04/09/2020

(This form should appear at the end of each thesis chapter/section/appendix submitted as a manuscript/ publication or collected as an appendix at the end of the thesis)

MASSEY UNIVERSITY
GRADUATE RESEARCH SCHOOL

STATEMENT OF CONTRIBUTION
DOCTORATE WITH PUBLICATIONS/MANUSCRIPTS

We, the candidate and the candidate's Primary Supervisor, certify that all co-authors have consented to their work being included in the thesis and they have accepted the candidate's contribution as indicated below in the *Statement of Originality*.

Name of candidate:	Sibghat Ullah
Name/title of Primary Supervisor:	Associate Professor Julian Jang-Jaccard
Name of Research Output and full reference:	
Bazai, Sibghat Ullah, Julian Jang-Jaccard, and Xuyun Zhang. "Scalable big data privacy with MapReduce." In Encyclopedia of big data technologies, pp. 1454-1462. Springer, Springer Nature, 2019.	
In which Chapter is the Manuscript /Published work:	
Please indicate:	
• The percentage of the manuscript/Published Work that was contributed by the candidate:	65
and	
• Describe the contribution that the candidate has made to the Manuscript/Published Work:	
Conceptualization, methodology, validation, formal analysis, investigation, original draft preparation, and visualization.	
For manuscripts intended for publication please indicate target journal:	
N/A	
Candidate's Signature:	Sibghat Ullah Bazai Digitally signed by Sibghat Ullah Bazai Date: 2020.09.03 18:32:43 +12'00'
Date:	03/09/2020
Primary Supervisor's Signature:	Julian Jang-Jaccard Digitally signed by Julian Jang-Jaccard Date: 2020.09.05 15:39:49 +12'00'
Date:	05/09/2020

(This form should appear at the end of each thesis chapter/section/appendix submitted as a manuscript/ publication or collected as an appendix at the end of the thesis)

MASSEY UNIVERSITY
GRADUATE RESEARCH SCHOOL

STATEMENT OF CONTRIBUTION
DOCTORATE WITH PUBLICATIONS/MANUSCRIPTS

We, the candidate and the candidate's Primary Supervisor, certify that all co-authors have consented to their work being included in the thesis and they have accepted the candidate's contribution as indicated below in the *Statement of Originality*.

Name of candidate:	Sibghat Ullah
Name/title of Primary Supervisor:	Associate Professor Julian Jang-Jaccard
Name of Research Output and full reference:	
Bazai, Sibghat Ullah, and Julian Jang-Jaccard. "SparkDA: RDD-Based High-Performance Data Anonymization Technique for Spark Platform." In International Conference on Network and System Security, pp. 646-662. Springer, Cham, 2019.	
In which Chapter is the Manuscript /Published work:	Chapter 4
Please indicate:	
• The percentage of the manuscript/Published Work that was contributed by the candidate:	70
and	
• Describe the contribution that the candidate has made to the Manuscript/Published Work:	
Conceptualization, methodology, validation, formal analysis, investigation, original draft preparation, visualization, and responding to reviewers' comments	
For manuscripts intended for publication please indicate target journal:	
Submitted to journal of MDPI electronic (Currently under review)	
Candidate's Signature:	Sibghat Ullah Bazai Digitally signed by Sibghat Ullah Bazai Date: 2020.09.03 18:38:43 +12'00'
Date:	03/09/2020
Primary Supervisor's Signature:	Julian Jang-Jaccard Digitally signed by Julian Jang-Jaccard Date: 2020.09.05 15:40:42 +12'00'
Date:	05/09/2020

(This form should appear at the end of each thesis chapter/section/appendix submitted as a manuscript/ publication or collected as an appendix at the end of the thesis)

MASSEY UNIVERSITY
GRADUATE RESEARCH SCHOOL

STATEMENT OF CONTRIBUTION
DOCTORATE WITH PUBLICATIONS/MANUSCRIPTS

We, the candidate and the candidate's Primary Supervisor, certify that all co-authors have consented to their work being included in the thesis and they have accepted the candidate's contribution as indicated below in the *Statement of Originality*.

Name of candidate:	Sibghat Ullah
Name/title of Primary Supervisor:	Associate Professor Julian Jang-Jaccard
Name of Research Output and full reference:	
Bazai, S. U., and Jang-Jaccard, J. In-memory Data Anonymization using Scalable and High Performance RDD Design. Submitted to journal of MDPI Electronics (Under review)	
In which Chapter is the Manuscript /Published work:	Chapter 4
Please indicate:	
• The percentage of the manuscript/Published Work that was contributed by the candidate:	70
and	
• Describe the contribution that the candidate has made to the Manuscript/Published Work:	
Conceptualization, methodology, validation, formal analysis, investigation, original draft preparation, visualization, and responding to reviewers' comments	
For manuscripts intended for publication please indicate target journal:	
N/A	
Candidate's Signature:	Sibghat Ullah Bazai Digitally signed by Sibghat Ullah Bazai Date: 2020.09.03 18:36:55 +12'00'
Date:	03/09/2020
Primary Supervisor's Signature:	Julian Jang-Jaccard Digitally signed by Julian Jang-Jaccard Date: 2020.09.05 15:41:00 +12'00'
Date:	05/09/2020

(This form should appear at the end of each thesis chapter/section/appendix submitted as a manuscript/ publication or collected as an appendix at the end of the thesis)

MASSEY UNIVERSITY
GRADUATE RESEARCH SCHOOL

STATEMENT OF CONTRIBUTION
DOCTORATE WITH PUBLICATIONS/MANUSCRIPTS

We, the candidate and the candidate's Primary Supervisor, certify that all co-authors have consented to their work being included in the thesis and they have accepted the candidate's contribution as indicated below in the *Statement of Originality*.

Name of candidate:	Sibghat Ullah
Name/title of Primary Supervisor:	Associate Professor Julian Jang-Jaccard
Name of Research Output and full reference:	
Bazai, S. U., Jang-Jaccard, J. and Alavizadeh, H. A Generic Approach for Subtree-Based Data Anonymization for Apache Spark. Submitted to journal of IEEE Access (Under review)	
In which Chapter is the Manuscript /Published work:	Chapter 5
Please indicate:	
• The percentage of the manuscript/Published Work that was contributed by the candidate:	70
and	
• Describe the contribution that the candidate has made to the Manuscript/Published Work:	
Conceptualization, methodology, validation, formal analysis, investigation, original draft preparation, visualization, and responding to reviewers' comments	
For manuscripts intended for publication please indicate target journal:	
N/A	
Candidate's Signature:	Sibghat Ullah Bazai Digitally signed by Sibghat Ullah Bazai Date: 2020.09.03 18:35:36 +12'00'
Date:	03/09/2020
Primary Supervisor's Signature:	Julian Jang-Jaccard Digitally signed by Julian Jang-Jaccard Date: 2020.09.05 15:41:15 +12'00'
Date:	05/09/2020

(This form should appear at the end of each thesis chapter/section/appendix submitted as a manuscript/ publication or collected as an appendix at the end of the thesis)

MASSEY UNIVERSITY
GRADUATE RESEARCH SCHOOL

STATEMENT OF CONTRIBUTION
DOCTORATE WITH PUBLICATIONS/MANUSCRIPTS

We, the candidate and the candidate's Primary Supervisor, certify that all co-authors have consented to their work being included in the thesis and they have accepted the candidate's contribution as indicated below in the *Statement of Originality*.

Name of candidate:	Sibghat Ullah
Name/title of Primary Supervisor:	Associate Professor Julian Jang-Jaccard
Name of Research Output and full reference:	
Bazai, S. U., Jang-Jaccard, J. and Alavizadeh, H. A Novel Hybrid Approach for Multi-dimensional Data Anonymization for Apache Spark. Transactions on Privacy and Security (Under Review)	
In which Chapter is the Manuscript /Published work:	Chapter 6
Please indicate:	
• The percentage of the manuscript/Published Work that was contributed by the candidate:	70
and	
• Describe the contribution that the candidate has made to the Manuscript/Published Work:	
Conceptualization, methodology, validation, formal analysis, investigation, original draft preparation, visualization, and responding to reviewers' comments	
For manuscripts intended for publication please indicate target journal:	
N/A	
Candidate's Signature:	Sibghat Ullah Digitally signed by Sibghat Ullah Date: 2020.11.30 12:24:18 +13'00'
Date:	30/11/2020
Primary Supervisor's Signature:	Julian Jang-Jaccard Digitally signed by Julian Jang-Jaccard Date: 2020.11.30 16:50:48 +13'00'
Date:	30/11/2020

(This form should appear at the end of each thesis chapter/section/appendix submitted as a manuscript/ publication or collected as an appendix at the end of the thesis)