POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Integration of Differential Privacy Mechanism to Map-Reduce Platform for Preserving Privacy in Cloud Environments

MELISSA VOSOUGH TEHRANI
Département de génie informatique et génie logiciel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie informatique

Mai 2020

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Ce mémoire intitulé :

Integration of Differential Privacy Mechanism to Map-Reduce Platform for Preserving Privacy in Cloud Environments

présenté par **Melissa VOSOUGH TEHRANI**

en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*

a été dûment accepté par le jury d'examen constitué de :

Michel DESMARAIS, président

John MULLINS, membre et directeur de recherche

Foutse KHOMH, membre

DEDICATION

To anyone, who tries to make the world a better place . . .

ACKNOWLEDGEMENTS

I like to thank Prof. John Mullins of the Computer and Software Engineering department as my thesis supervisor.

I also like to express my deep and sincere gratitude to Emilio Rivera and Sardaouna Hamadou for conducting me through this research work.

I would like to extend my gratitude to the members of the jury, Dr. Foutse Khomh and Dr. Michel Desmarais who reviewed my thesis for their valuable feedbacks.

Finally, I would like to thank everyone at the Computer and Software Engineering department who contributed to the success of my thesis.

RÉSUMÉ

Le cloud computing peut être désigné comme utilisant les capacités de ressources matérielles et logicielles basées sur Internet; C'est la tendance de la dernière décennie dans le monde numérique d'aujourd'hui, de plus en plus rapide. Cela a changé le monde qui nous entoure. L'utilisation du cloud est devenue une norme et les utilisateurs transfèrent leurs données vers le cloud à mesure que les données grossissent et qu'il est nécessaire d'accéder aux données à partir de nombreux appareils.

Des tonnes de données sont créées chaque jour et toutes les organisations, des instituts scientifiques aux entreprises industrielles, ont pour objectif d'analyser les données et d'en extraire les schémas afin d'améliorer leurs services ou à d'autres fins.

Dans l'intervalle, les sociétés d'analyse de données utilisent les informations de millions de personnes et il est de plus en plus nécessaire de garantir la protection de leurs données. Des techniques d'ingénierie sociale aux attaques techniques malveillantes, les données risquent toujours de fuir et nous devrions proposer des solutions pour protéger les données des individus.

Dans cette thèse, nous présentons «Parmanix», une plateforme de protection de la confidentialité pour l'analyse de données. Il est basé sur le système MapReduce et fournit des garanties de confidentialité pour les données sensibles dans les calculs distribués sur des données sensibles. Sur cette plate-forme, les fournisseurs de données définissent la politique de sécurité de leurs données. Le fournisseur de calcul peut écrire du code Mapper non approuvé et utiliser l'un des réducteurs de confiance déjà définis dans Parmanix. Comme le système garantit une surcharge acceptable, il n'y aura aucune fuite de données individuelles lors des calculs de la plate-forme.

ABSTRACT

Cloud computing can be referred to as using the capabilities of hardware and software resources that are based on the Internet; It is the trend of the past decade growing among today's digital world at a fast pace. It has changed the world around us. Using the cloud has become a norm and people are moving their data to the cloud since data is getting bigger and there is the need to access the data from many devices. Tones of data are creating every day and all the organizations, from science institutes to industrial companies aim to analyze the data and extract the patterns within them to improve their services or for other purposes.

In between, information of millions of people is getting used by data analytic companies and there is an increasing need to guarantee the protection of their data. From social engineering techniques to malicious technical attacks, the data is always at the risk of leakage and we should propose solutions to keep an individual's data protected.

In this thesis, we present "Parmanix", a privacy preserve module for data analytics. It is based on the MapReduce system and provides privacy guarantees for sensitive data in distributed computations on sensitive data. With this module, data providers define the security policy for their data, and computation provider can write untrusted Mapper code and use one of the trusted Reducers that we have already defined within Parmanix. As system guarantees with an acceptable amount of overhead, there would be no leakage of individual's data through the platform computations.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

FOIL	Freedom of Information Law
GIC	Group Insurance Commission
HDFS	Hadoop Distributed File System
IMDB	Internet Movie Database
RPC	Remote Procedure Call libraries
PINQ	Privacy Integrated Queries
PM	Partition Map

LIST OF APPENDICES

CHAPTER 1 INTRODUCTION

Cloud computing, one of the biggest trends of technology in recent years, has faced rapid development. Cloud computing today is a mature market that conducted several large companies to build efficient cloud environments.

Data analytics and cloud computing essentially go hand in hand and as clouds become more inclusive, data analytics in cloud platforms continue to grow as well. New methods and new platforms of data analytics bring innovative improvements for so many areas such as e-commerce, healthcare, etc. and adds so many positive possibilities for businesses. However, there are just as many new privacy concerns being created.

Collecting and managing data creates significant privacy risks that make "security" and "privacy" of information as a concerning issue. At the time of processing, a trusted privacy model is required in order to prevent data leakage and defense against outside attacks. Even in the storage phase, prevention against possible security threats is needed.

Big data privacy can be preserved by different approaches. The main approaches categorize into three groups [1]; Data encryption, Anonymization techniques and, Noise-based approaches. The first two help to hide the sensitive data, however, due to the presence of many re-identification techniques, they cannot guarantee privacy [2].

On the other hand, the noise-based solutions have shown better utility for statistical data analytics [3].

In this research work, we have studied a noise-based privacy algorithm, "Differential Privacy", and implemented it in "Hadoop" a data analytic platform. Differential Privacy promises to overcome the privacy flaws of encryption and anonymization-based solutions. Our privacy platform, which we call it "Parmanix", offers a solution where the risks and costs associated with privacy issues are on the rise.

Differential Privacy is a mathematical definition for the privacy loss and measures how effective it would be to use random noise insertion as a particular privacy technique at protecting the privacy of individuals' information within a dataset [4].

There are privacy parameters to determine the amount of noise or perturbation that should be applied to the dataset to achieve the desired level of privacy. In this algorithm, privacy is quantified by how statistically indistinguishable are the privacy-preserved outputs of the dataset [5].

Hadoop is a powerful parallel data processing model and is widely used due to its ability to solve many of large-scale computing problems [6]. However, this platform does not provide any privacy solution and still lacks in this aspect; In fact, the privacy depends on the dataset itself whether it is anonymized or encrypted. Hadoop's various modules have been developed separately over time and eventually, the security and privacy of this platform have not been under development until recently [7].

"Parmanix" by incorporating the Laplacian Differential Privacy algorithm to the Hadoop MapReduce platform, guarantees the privacy of individuals' information in a dataset that is being used for data analytic purposes while it does not burden extra overhead that other privacy solutions such as cryptography algorithms impose to the computation. In our proposed platform, users without security expertise can perform computations on the datasets and no extra knowledge is required which makes Parmanix a flexible tool.

1.1 Research Objectives

This research work aims at studying privacy solutions in data analytic platforms. Considering today's data analytic world demand, we inspect factors such as privacy guarantee in different scenarios, performance, the complexity of use for both data provider and computation provider and based on these factors we selected the best privacy approach. Moreover, we selected Hadoop and implemented Differential Privacy (which is our selected privacy solution) and developed a new platform, Parmanix. We selected Hadoop as it is an open-source and a well-known data analytic platform used by many businesses and enterprises around the world [8].

Parmanix makes changes in the regular procedure of Map-Reduce which is a module inside the Hadoop platform to implement differential privacy which unlike its counterparts -such as anonymization technique- guarantees the privacy of individuals information in datasets.

To assess the effectiveness of our proposed platform, we conduct a series of experiments including investigation on the effect of each privacy parameter on the output results, performance and utility of Parmanix and finally assess it with Naive Bayes machine learning algorithm.

We address the following three research questions:

RQ1: Why is there a need for a privacy solution in Hadoop MapReduce and can noise-based solutions be practical to provide this privacy?

Hadoop is one of the widely used data analytic platforms all around the world and by big companies. However its security and privacy has not been developed as rapid as its use expansion. This has made it a challenge and as a result Hadoop lacks a consistent security model and there are a lot of privacy threats that need to be worked on.

Three main privacy approaches in data analytics include Data encryption, Anonymization, and Noise-based techniques [1]. Data encryption is an efficient means of preventing unauthorized access to data and helps to avoid exposure of data to breaches such as packet sniffing, the man in the middle, etc [9].

Encryption technology adds limitations to the application functionality in using data, as it uses the computational-resources intensively. This limitation gets more challenging when it has to deal with the volume, velocity, and variety of big data [10].

Using encryption in data analyzing platforms such as Hadoop means putting extra pressure on the system's resources and a considerable reduction in performance. The data must be decrypted in order to be processed in MapReduce, and this not only decreases performance considerably for large datasets, but it will also put the data in the risk of leakage (in MapReduce phase) especially with the fact that Hadoop spreads data across a large number of nodes [11]

As a result, while encryption increases the security for Hadoop, it is not necessarily guarantee it and is not able to provide acceptable performance for huge amounts of data.

The other privacy technique is data anonymization wherein the information that discloses the identity is removed from datasets; this helps to keep people who are defined by the information remain unknown [12]. While the anonymization technique increases privacy, it can not guarantee privacy in every scenario as we show throughout an experiment where the dataset has been anonymized yet it reveals an individual's data by some queries. We showed in the same scenario, with the same dataset and the same queries, differential privacy as a noise-based approach, keeps the promise of privacy.

As a result, we chose Differential Privacy, which is a noise-base solution and do not impose the limitations that mentioned for previous solutions. It calculates the proper amount of noise through the privacy parameters that it receives and by applying it to the available output data stops the attacker to conclude anything with 100% confidence [13].

RQ2: Can we design and implement a noise-based privacy solution in Hadoop as a data analytic platform in order to guarantee privacy while imposing an acceptable overhead?

We selected Differential Privacy as a noise-based solution and implemented it in Hadoop MapReduce. This new module which we called it Parmanix, has shown privacy guarantee for the experiments that we designed depending on the privacy parameters that user chose. We have depicted this with experiments. To evaluate the performance, we tested our module with three different dataset sizes. The overhead changed from 12% to 17% from smallest to largest dataset.

RQ3: How Differential Privacy affects the accuracy of the output and how to make a balance between privacy and accuracy of the output results?

One of our challenges in this research work is a trade-off between the privacy of the result and accuracy. Parmanix in any situation keeps its promise of privacy; However, in some scenarios, this privacy comes with the cost of accuracy.

The philosophy of data analytics is to extract meaningful results and patterns from data. So in order to know how we can keep the balance between privacy and accuracy, we scrutinized each of the parameters that are used to calculate noise in the Differential Privacy algorithm. These parameters include Epsilon and sensitivity, which we have described them in chapter 2.

Epsilon must be specified by the computation provider. Sensitivity, on the other hand, will be perceived by the dataset itself. The higher the epsilon is, the lower is the amount of noise. While to a dataset with higher sensitivity, more noise will be added. Depending on the importance of the data in that dataset, what patterns the data owner needs to extract for analytics, etc, these parameters must be specified to keep the balance.

1.2 Thesis Plan

The rest of this thesis is organized as follows:

In chapter 2 we overview the preliminary concepts and mathematical point of view of the used methods that are related to our work and are necessary to understand Parmanix architecture.

We reviewed the related literature, in chapter 3 and did extra discussion over our platform and algorithm. Chapter 4 outlines the methodology and architecture of our privacy preserve platform.

The results of our case studies that aimed to evaluate the effect of privacy parameters on the trade-off between privacy and accuracy of Parmanix output, has been discussed in chapter 5. In this chapter, we also scrutinized the effectiveness of our proposed solution for different datasets in different scenarios and made a comparison between the performance of Parmanix and original Hadoop MapReduce.

CHAPTER 2 BACKGROUND

2.1 Hadoop MapReduce

Parmanix is based on Hadoop Map-Reduce. Hadoop is an open-source cloud computing and big data framework based on Java that supports the processing of large datasets in a distributed computing environment. Hadoop allows running the applications on systems that have thousands of nodes and each node processes thousands of terabytes of data. We had two criteria to choose from Hadoop MapReduce.

Criterion 1: Accessibility

Since Hadoop Map-Reduce is an open-source project, its source code, bugs and patches are available online. Hadoop is one of the most popular open-source data analytic platforms.

Criterion 2: Extensive Use

Over the last few years, interest in data science has grown tremendously and Hadoop was one of the main tools used for this purpose. Being widely used increases the importance of privacy solutions for the mentioned platform and increase the interest over our solution as many companies are using Hadoop for their data analytic works.//

To understand the architecture of Parmanix, it is necessary to know the architecture of the platforms it is residing in and the changes we made on their architecture. Hadoop has 4 core components: Hadoop Common, YARN, Hadoop Distributed File System (HDFS) and Map-Reduce.

Hadoop Common is the set of utilities and libraries that other modules of Hadoop use. Modules such as Hive, HBase, etc. As an example, if Hive wants to access HDFS it will use Java archives that have been stored in Hadoop Common.

Hadoop YARN is the resource manager and job scheduler in the Hadoop distributed processing framework. YARN's main responsibility is allocating system resources to the various applications running in a Hadoop cluster and scheduling tasks to be executed on different cluster nodes. As an example, at Yahoo company, 80,000 jobs a day were loaded on the most heavily used Hadoop clusters while with the help of YARN, they increased it to a load of 125,000 jobs a day. This is an approximate 50 percent increase. This means in a certain amount of time, more jobs have been done.

In Parmanix, Hadoop common and Yarn will work, and we have not made any changes in their source code.

Hadoop Distributed File System or HDFS, is the file system specifically designed for Hadoop. It runs on clusters of commodity hardware. This distributed file system supports fast data transfer rates among nodes, and it is highly scalable and fault-tolerant. The main reason that HDFS has shown capable of fault management is that its component creates several replicas of the data blocks. These data blocks will be distributed across various clusters for reliable and quick data access. Eventually, if any failure happens, it restarts failed subtasks and allows the system to continue operating uninterrupted at times of node failure. Hence, it can manage the data of the MapReduce applications.

NameNode, DataNode and Secondary NameNode are 3 important components of HDFS. HDFS has a Master/Slave architecture; the NameNode acts as the master node; It keeps track of the storage cluster, manages the file system namespace operations like opening, closing, and renaming files and directories, determines the mapping of blocks to the DataNodes and regulates access to the files by users.

DataNode acts as a slave node and sums up the various systems within a Hadoop cluster. It is responsible for serving file systems to read and write requests. It also performs block creation, deletion, and replication upon demand from the Namenode (Master) The architecture of HDFS has been shown in Figure 2.1.

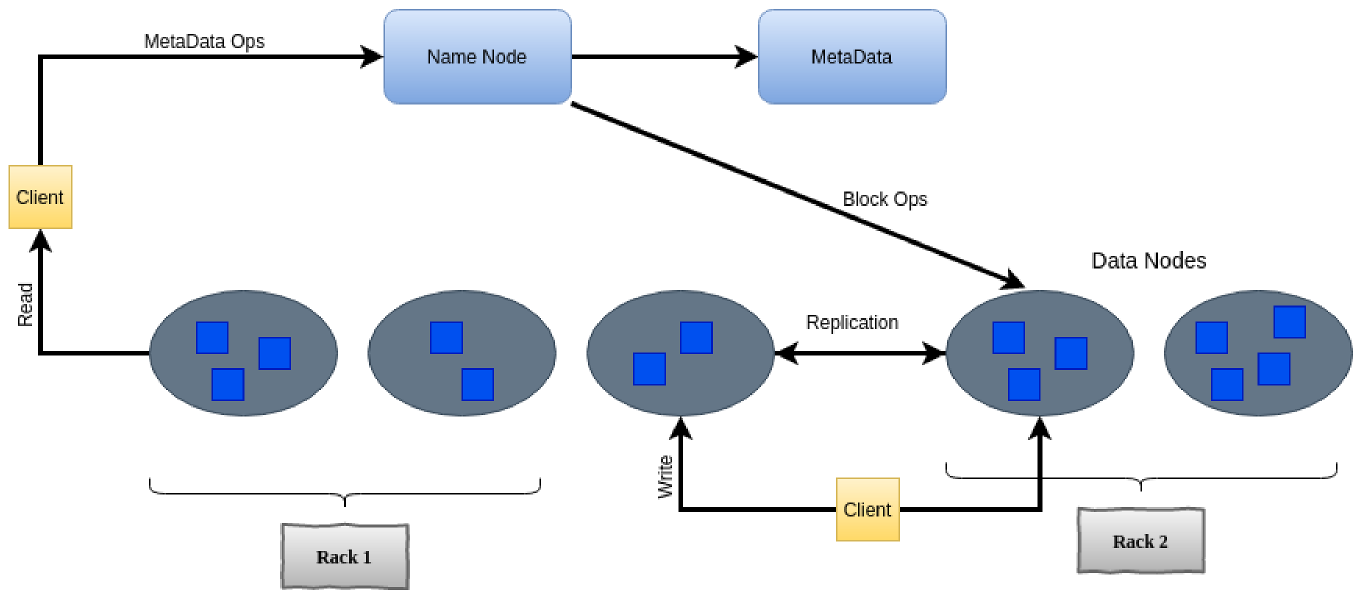


Figure 2.1 Hadoop Distributed File System Architecture

The fourth main component of Hadoop is **MapReduce**; It is a programming model within the Hadoop framework used to access big data stored in the HDFS.

MapReduce allows non-expert users to easily code their queries using Java rather than SQL and run analytical tasks over big data. So, no background in databases is required and with Java they can do the work. This independence from databases has increased query processing efficiency significantly and has made Hadoop MapReduce to be used by a large number of developers. In this framework the developer does not need to get involved in parallelization issues; instead, they can only focus on their computation problem and write parallel processing programs.

Parmanix is running as a module on MapReduce. As the name states MapReduce includes two main data processing functions: Map and Reduce.

The procedure is as input files will be saved on Hadoop distributed file system, input data will be partitioned into fixed-sized blocks called chunks and a MapReduce computation will read them. Each chunk will be assigned to a Mapper.

These parallel Map tasks perform the computation on the input chunk and produce intermediate output as a collection of <key, value> pairs. These pairs are shuffled across different reduce tasks based on the <key, value>.

The Map function takes input from the disk as <key, value> pairs, processes them, and produces an output containing a new set of intermediate <key, value> pairs. For example, a file has 200 records to be processed, then 200 Mappers can run together while each Mapper processes one of the records. Or maybe 100 Mappers can run together while each Mapper process 2 of the records

Based on the size of the data to be processed and the memory block available on each Mapper server, the Hadoop framework decides how many Mappers should be used.

When all of the Mappers completed processing, the framework shuffles and sorts the results. Note that a Reducer cannot start its work while a Mapper is still in progress. After the Mapper finished the work, now it is the Reduce function stage to take these intermediate <key, value> pairs as their inputs.

All the map output values that have the same key are assigned to a single Reducer, which then aggregates the values for that key. Each of the Reduce tasks accepts only one key at a time and process data for that key and outputs the results as <key, value> pairs. In the final step, the values will be combined with distinct keys in the reduce phase and produce the final result as an output file. The general steps of a MapReduce task have been depicted in Figure 2.2

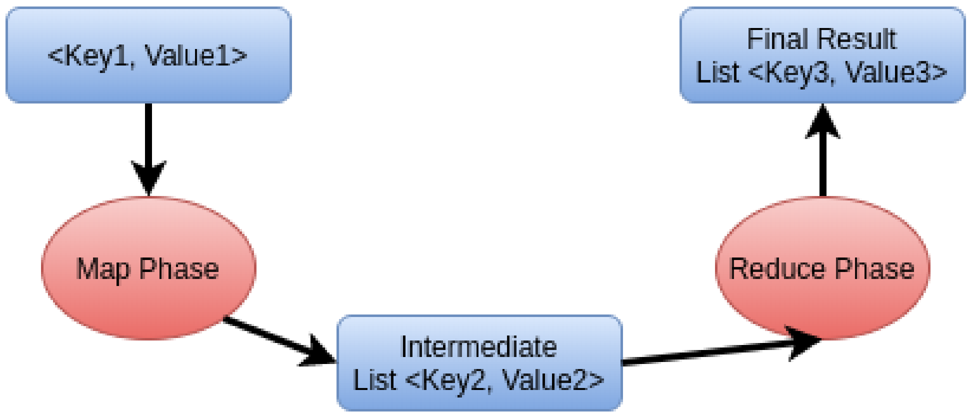

<Key1, Value1>
Map Phase
Intermediate List <Key2, Value2>
Reduce Phase
Final Result List <Key3, Value3>

Figure 2.2 MapReduce Steps

Same as HDFS which has a Master/Slave architecture, Hadoop's MapReduce architecture is based on a Master/Slave communication model too. This architecture consists of one JobTracker as master and many TaskTrackers as workers or slaves.

JobTracker creates and runs the job. It runs on the NameNode and allocates the job to Task-Trackers. Tracking resource availability, managing tasks life cycle, tracking tasks progress, fault tolerance, etc are among the duties of the JobTracker.

TaskTracker runs on DataNode and executes the tasks and report the status of the task to JobTracker. It has the function of following the orders of the job tracker. Periodically it updates the job tracker with its progress status.

To get more details about JobTracker and TaskTracker in the MapReduce process, it must be stated that the jobs submitted by the user will be received by JobTracker as it is the point of interaction between users and the MapReduce framework.

When a MapReduce job is submitted, Jobtracker breaks it down into Map and Reduce tasks, puts it in a queue of pending jobs and executes them on a FIFO or First-Come-First-Served basis. It assigns the tasks to TaskTrackers and permanently monitors TaskTracker progress. TaskTracker executes tasks upon instruction from the Master (JobTacker) and handle data motion between the Map and Reduce stages.

Each Task Tracker has a fixed number of Map and Reduce task slots. By completing all the tasks, JobTracker notifies the user that the job has been completed.

Figure 2.3 shows each MapReduce stage with an input that has been stored n HDFS. As can be seen the input breaks into chunks (split), and they assigned to each Mapper (Map), the outputs go into the Combiner (Combine) and produce the final result (Reduce) which will be stored in HDFS.

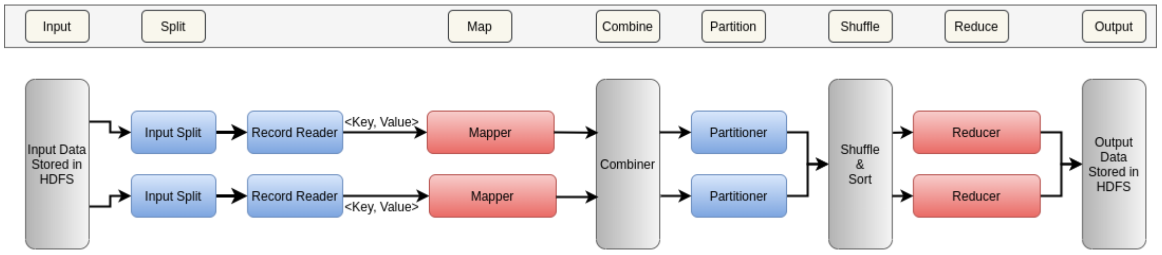


Figure 2.3 MapReduce Execution

A real-world example can depict these processes. Imagine Amazon company wants to know its total sale for each city of Canada in 2018. To do this, they should count the number of orders submitted for each city. The number of records in the related dataset is significantly high. If this computation is done by traditional solutions such as SQL, it will take hours and maybe days! However using the MapReduce technique, will solve this issue.

To go through this computation, the first step is writing the Mapper code by the computation provider. In the Mapper, he should define "Key" and "Value". Table 2.1 is one record of this dataset:

Table 2.1 One Record Of The Dataset

Number	Date	City	Product Name	Price	Number Of Orders	Total Sale Amount
1	2018/05/10	Vancouver	Adidas T-shirt	150	2	300

The computation provider will define a key as "City" name and the "Value" as "Total Sale Amount". The whole data breaks into chunks. The chunks will be assigned to the Mappers. Assume we have 12 Mappers and they work on the data in parallel at the same time with small fractions of data that they have (Figure 2.4). Each Mapper gives out a record of data containing the name of the city and its sales and writes it on the index card; The same happens for other Mappers and they give out the records until all the Mappers end up having a series of blocks as their output. The Mappers job is over.

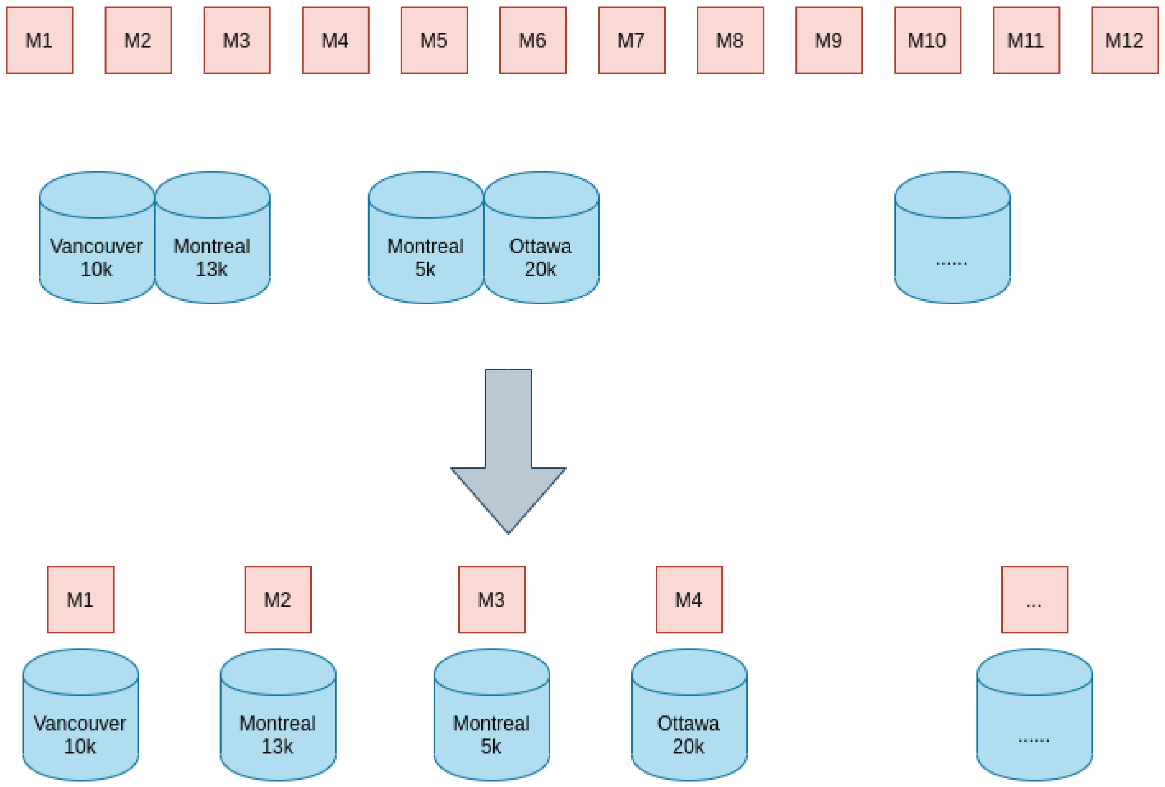


Figure 2.4 Data Breaks Into Smaller Chunks And Get Assigned To The Mappers

The partitioner is an intermediate phase between Map and Reduce. It sends intermediate <key, value> pairs to the Reducer. Partitioner ensures that all <key, value> pairs are grouped by their key on a single Reducer and results in a roughly balanced load across all the Reducers. Finally, the Reducers aggregates data according to keys and sends the <key, value> pairs to the output (Figure 2.5). This output will be translated to file format and we will have the total sale of Amazon in each Canadian city.

Note that In Parmanix we did not touch the general procedure of MapReduce, however, the details have been changed and limitations have been applied.

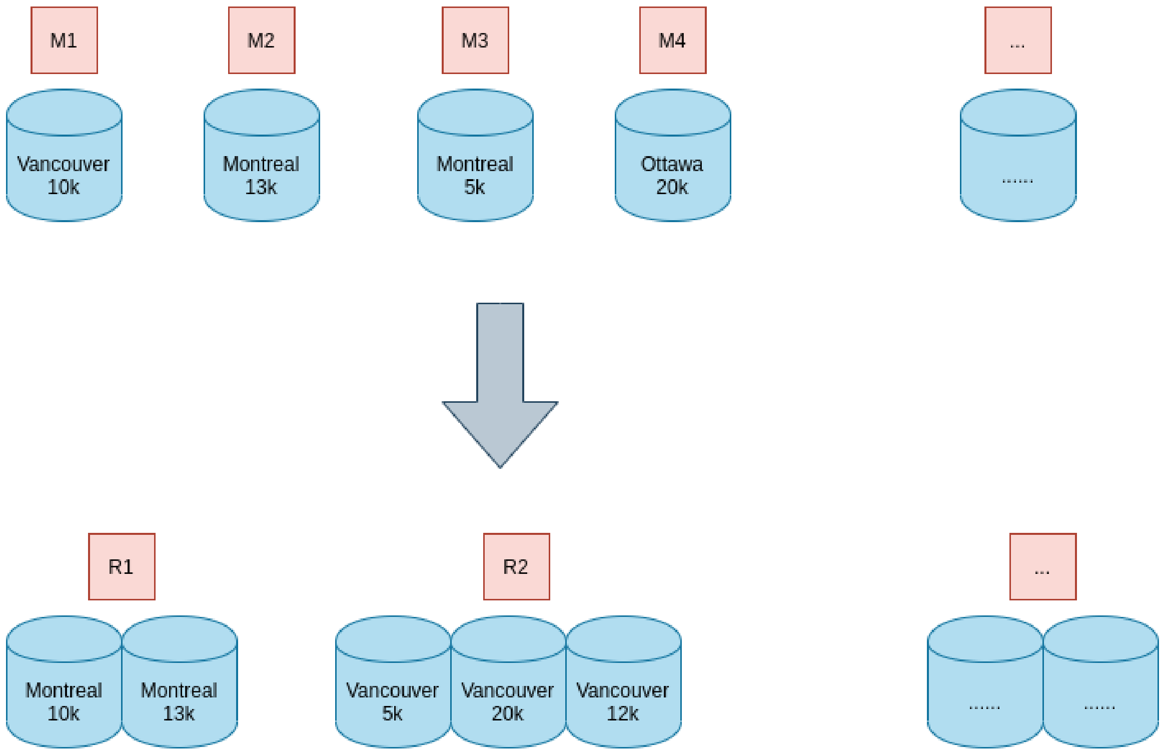


Figure 2.5 Reducers Receive The Key And Values Produced By Mappers

2.2 Combiner

In the MapReduce process, there are two main stages which are Map and Reduce which previously described them in detail. Aside from these two main stages, there are two intermediate stages.

Partition which is the process that gets the produced <key, value> pairs from Mapper and translates the pairs to another set of <key, value> pairs to feed into the Reducer. It decides how the data must be presented to the Reducer and assigns it to a particular Reducer.

The second intermediate stage is Combine. **combine** is not mandatory, and it can be enabled or disabled according to the choice of user. We can define the combiner as a Reducer that runs individually on each Mapper server. Combiner reduces the time taken for data transfer between Mapper and Reducer; in fact, it reduces the data on each Mapper to a simplified form and passes it downstream; as a result, there would be fewer data to work on which this makes shuffling and sorting easier.

As an example [14], assume that Hadoop runs three Mappers. Mapper 1, Mapper 2 and Mapper 3. The input of Mapper contains the value as a record of the log file and the key could be a text string such as "file name". The Mapper processes each of these records of the log file and produces key value pairs. Mapper output has been shown in table 2.2.

Table 2.2 Mapper Outputs In The Combiner Example

Mapper1	Mapper2	Mapper3
<Exc A, 1>	<Exc B, 1>	<Exc A, 1>
<Exc B, 1>	<Exc B, 1>	<Exc C, 1>
<Exc A, 1>	<Exc A, 1>	<Exc A, 1>
<Exc C, 1>	<Exc A, 1>	<Exc B, 1>
<Exc A, 1>		<Exc A, 1>

For each Mapper a combiner is running: Combiner 1, Combiner 2 and Combiner 3. The output of the Mappers will be the input of the combiners. Same as the reducer, these combiners calculate the count of each exception. The input from Mapper 1 to Combiner 1 will be:

<Exception A, 1>, <Exception B, 1>, <Exception A, 1>, <Exception C, 1>, <Exception A, 1>

The same applies to other Combiners. The output of Combiners has been shown in Table 2.3.

Table 2.3 Combiner Outputs In The Combiner Example

Combiner1	Combiner2	Combiner3
<Exc A, 3>	<Exc A, 2>	<Exc A, 3>
<Exc B, 1>	<Exc B, 2>	<Exc B,1>
<Exc C, 1>		<Exc C, 1>

In the next step, the partitioner sorts the data and allocates it from the combiners to the Reducers. Table 2.4 shows the input to the reducers.

Table 2.4 Reducer Inputs (With Combiner)

Reducer1	Reducer2	Reducer3
<Exc A> (3,2,3)	<Exc B> (1,2,1)	<Exc C> (1,1)

If there was no combiner involved the input to the reducers would be like Table 2.5.

This is a simple example, however, when there are terabytes of data involved, an enabled combiner has a significant improvement to the bandwidth. Finally, each of the reducers calculates the total count of the exceptions and the computation will be completed (Table 2.6)

The function and process of combiner has been shown in figure 2.6 in details.

Table 2.5 Reducer Inputs (Without Combiner)

Reducer1	Reducer2	Reducer3
<Exc A> (1,1,1,1,1,1,1,1)	<Exc B> (1,1,1,1)	<Exc C> (1,1)

Table 2.6 Final Results

Reducer1	Reducer2	Reducer3
<Exc A, 9>	<Exc B, 4>	<Exc C, 2>

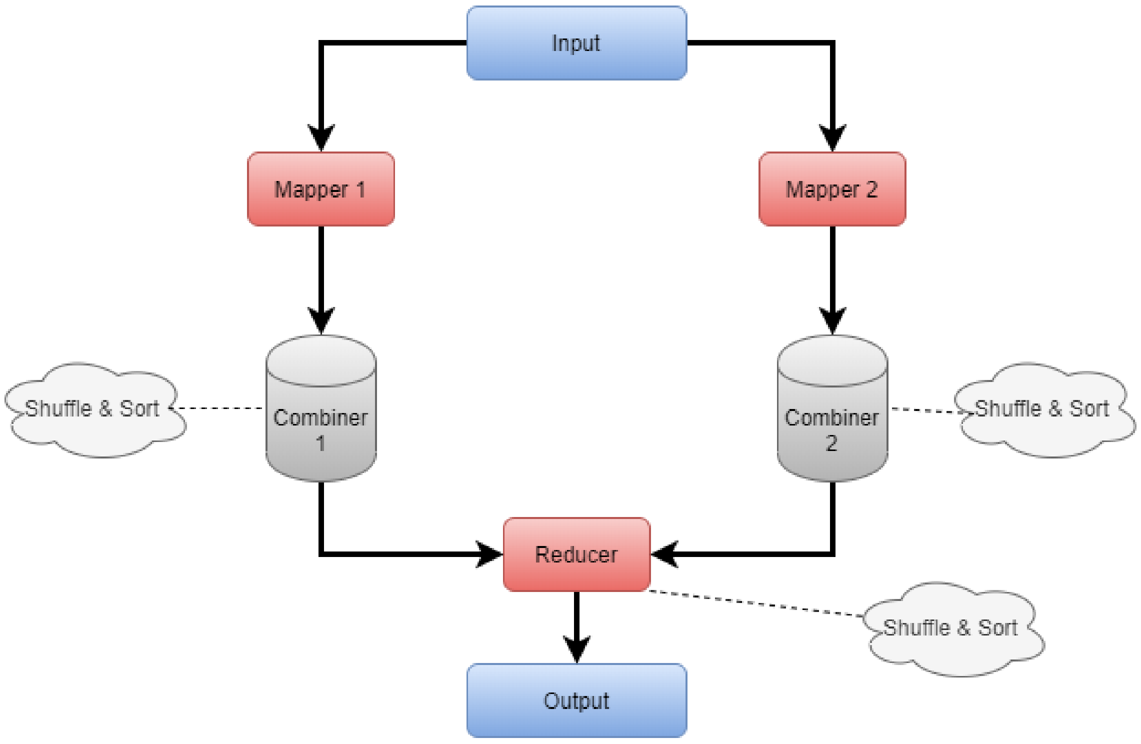


Figure 2.6 Combiner Function in MapReduce

2.3 Differential Privacy

The objective of Parmanix module is to preserve privacy in large scale computations that include datasets from various resources and include individual's data. Parmanix is using the Differential Privacy algorithm to get close to this goal.

A mechanism is differentially private if every output is produced with a similar probability whether any given input is included or not [15].

Every day, every individual produces data including financial data, health data, data produced from web search or social networks and this list goes on. These data can be used for different purposes, finding a statistical correlation, publishing aggregate statistics or data mining on customer's data to improve the company's strategies. Many sensitive data exists in these datasets. The question is how we can promise the privacy of these sensitive data? How can we guarantee the privacy of individuals whom their data is in the datasets?

In this situation, the main concept is to promise that a person by accessing the output of data computation does not be enabled to learn anything about an individual that could not be learned without access to that output.

To demonstrate Differential Privacy better, first, we go with an example. We assume a company wants to know the correlation between smoking and having lung cancer and uses the health dataset to do this analysis. This dataset contains information about people including their smoking history and cancer history.

Table 2.7 Records Of A Health Dataset

Name	age	Smoker	Lunge Cancer
David Smith	32	Y	Y
Samantha Roth	34	N	N
Margareth Pena	61	N	N
Samuel McGee	45	Y	N
Martha Christensen	28	Y	N

Assume the outcome indicate a correlation of 20% chance of having lunge cancer in future for Samuel, based on his health information. If we want to keep our promise of privacy, instead of the output 20%, we can think about a randomized output. Formally said:

Having dataset D, the algorithm A probabilistically maps D to an object or event in outcome space. Or as [15] has defined, A is ϵ-differentially private if for all databases D and D' which differ in only one individual:

$P[A(D) = O] \leq e^{\epsilon}.P[A(D') = O]$

this must be true for all possible outputs O.This notion has been parametrized by epsilon. $P[A(D) = O]$ is the probability that when we run the process A on the database D, the output is O.

e^{ϵ} is the exponential function applied to the parameter and $\epsilon > 0$. If ϵ is very close to 0, then e^{ϵ} would be very close to 1, eventually the probabilities would be very similar. On the other hand, the bigger ϵ is, the more the probabilities can differ.

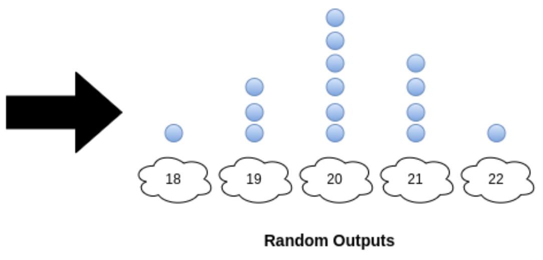
This definition is symmetrical which means you can replace D by D' and vice-versa, and the two databases will still differ in only one individual. So, we could replace it by:

$e^{-\epsilon}.P[A(D') = O] \leq P[A(D) = O] \leq e^{\epsilon}.P[A(D') = O]$

Back to our example, the real output for each record of dataset, under differential privacy should be randomized so a mechanism under a particular database will produce a particular set of distribution over a set of possible outcomes.

Differential privacy says that if we add or remove one of the rows of the dataset, this set of distributed outputs should be very close to the previous. Figure 2.7 depicts this.

Name	age	Smoker	Lunge Cancer
David Smith	32	Y	Y
Samantha Roth	34	N	N
Margareth Pena	61	N	N
Samuel McGee	45	Y	N
Martha Christensen	28	Y	N



Name	age	Smoker	Lunge Cancer
David Smith	32	Y	Y
Samantha Roth	34	N	N
Margareth Pena	61	N	N
~~Samuel McGee~~	~~45~~	~~Y~~	~~N~~
Martha Christensen	28	Y	N

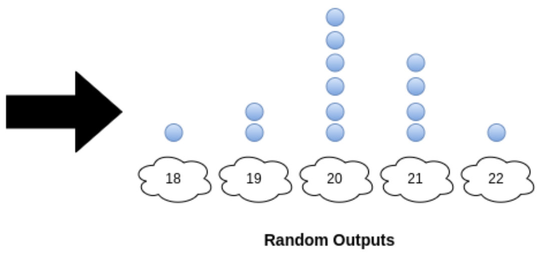


Figure 2.7 Probabilistic Output With/Without One Individual

Now a person might not participate in this study as she might be worry that this information sharing might affect her premium insurance cost. Whether she participates or she does not, the probability of the outcome results would be the same.

In [16] it is noted that differential privacy promises the outcome of a survey will stay the same, whether you participate in it or not.

Note that differential privacy is a property of a mechanism. It is not possible to take an outcome and state that it is differentially private. The process that has produced this outcome is differentially private.

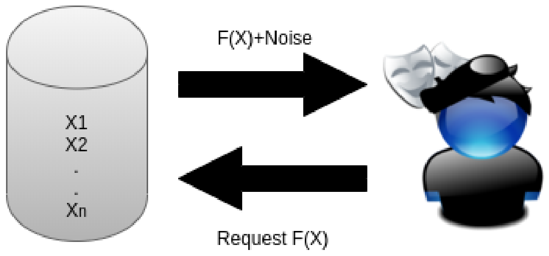


Figure 2.8 Differential Privacy Mechanism

2.3.1 Sensitivity

As mentioned to have the randomness we should have a set of distribution, so we add noise with a particular shape. The question is how much noise must be added and what is this particular shape?

"Sensitivity" is the factor that decides how much noise should be added. So we need to calculate the sensitivity of a function. By sensitivity, one means if we add or remove data from the dataset, how much the output would change? It measures how much one person can affect the output.

The sensitivity of a counting query that counts the number of dataset rows to satisfy a predicate, is 1. Because if we add or remove one person from a dataset, the maximum change that will occur would be 1.

The mathematical definition of sensitivity is

$$\begin{aligned} & f : D \to R^k \\ & \Delta f = \max_{D_1,D_2} \left\| f(D_1)) - f(D_2)) \right\|_1 \end{aligned} \tag{2.1}$$

while the datasets D_1 and D_2 are datasets that are different in only one item that exist in D_1 but not in D_2 .

So the first tool in differential privacy is noise addition. Calculating the noise to add, will come from a probability distribution that is called the Laplace Distribution (Figure 2.9). In fact “Differential privacy is a strong, mathematical definition of privacy in the context of statistical and machine learning analysis.” [17] This algorithm provides a mathematically provable guarantee of privacy against a wide range of privacy attacks such as attempts to learn individuals private information.

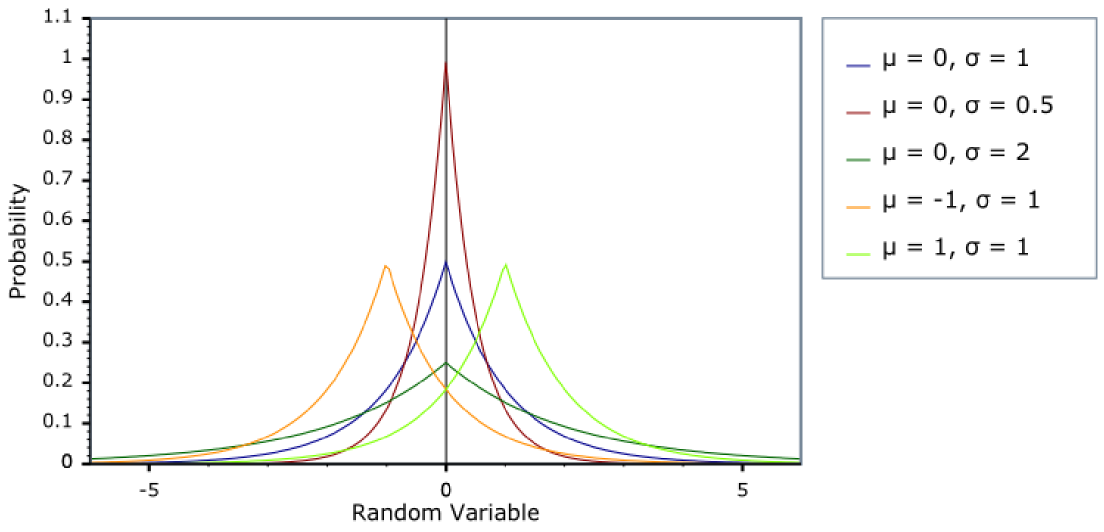


Figure 2.9 A Laplacian Diagram

In a counting query which is supposed to satisfy the predicate P, the sensitivity is 1 and the noise that should be added is $Lap(1/\epsilon)$.

Differential privacy allows for a deviation between two scenarios that individual X exists in the dataset and the scenario that X does not exist in the dataset. The privacy parameter “epsilon” is the parameter in differential privacy that quantifies the extent of this deviation; Epsilon or ϵ is, in fact, the privacy loss parameter that measures the effect of each individual’s information on the output of the analysis.

Choosing a value for ϵ expresses the amount of noise that is needed to provide a certain level of privacy protection. This directly affects the utility or accuracy that will be obtained from the analysis. The smaller the ϵ gets, the smaller deviation and therefore stronger privacy protection will be provided but the accuracy will decrease.

2.3.2 Quantifying Adversary’s Knowledge

We consider our privacy scenario from the adversary’s perspective. Assume that mechanism A is ϵ-differentially private. We run it on database D, and release the output to the adversary. Then, the adversary tries to figure out whether their target person is in D. No matter how much information the adversary has about the dataset, under Differential Privacy, the attacker can’t gain a lot of information about their target.

Now assume the adversary knows everything about the dataset except their target person.

Between the two options he has, he should determine which dataset is the real one; The dataset with their target in it (D_1), or the other dataset without the target in it (D_2).

In the adversary's model of the world, the actual database D can be either D_1 or D_2. They might have an initial suspicion that their target is in the dataset. We represent the suspicion that the attacker has about the existence of their target in the dataset with the probability, $P[D = D_1]$.

This probability can be anything between 0 and 1. For example 0.01 if they think it's very unlikely that the target be in the dataset, or it be 0.9 if the adversary's suspicion is strong and 0.5 if they have no idea whether the person information exists in the dataset.

Similarly, $P\left[D = D_2\right]$ would be the probability of their suspicion that their target is not in the dataset; Since there are only two options, $P[D = D_2] = 1 - P[D = D_1]$ Now, after running the computation, we will have output O. The question is how much information did the attacker gain by this output?

To answer this, we should look at how much their suspicion changed after seeing this output. So, we must compare $P[D = D_1]$ with the updated suspicion $P[D = D_1 | A(D) = O]$. This is now the attacker's updated model of the world after seeing O.

With Differential Privacy, the updated probability is never too far from the initial suspicion which lets us quantify this phenomenon. Whether the adversary has full background knowledge or has partial knowledge about the dataset the bounds still hold.

2.3.3 Composition

In the previous section, we explained quantify information gain in our Differential Privacy algorithm that we have implemented in Parmanix. In this section, we will prove that Differential Privacy and eventually our computations in Parmanix are composable. Composable means that when two algorithms A and B are ϵ-differentially private then publishing the result of both is 2ϵ-differentially private.

Assume C the algorithm which combines A and B:

$$C(D) = (A(D), B(D)) \tag{2.2}$$

The output of this algorithm should be a pair of outputs:

$$O = (O_A, O_B) \tag{2.3}$$

The two algorithms are independent and have their own randomness, hence the result of one does not have an impact on the result of the other. We can write

$$\begin{aligned} P[C(D_1) = O] &= P[A(D_1) = O_A] \cdot P[B(D_1) = O_B] \\ &\leq e^{2\epsilon} \cdot P[A(D_2) = O_A].P[B(D_2) = O_B] \\ &\leq e^{2\epsilon} \cdot P[C(D_2) = O] \end{aligned} \tag{2.4}$$

thus, C is 2ϵ-differentially private.

CHAPTER 3 LITERATURE REVIEW

3.1 Hadoop MapReduce Privacy Approaches

MapReduce is a programming system for the distributed processing of large-scale data. It provides an efficient and faults tolerant manner on cloud platforms. MapReduce is fundamentally different from the classical computation in the cloud environments and due to its massive parallel processing style, it leads to distinct design challenges for security and privacy requirements.

There are research works on securing this platform. Owen O'Malley et al. [18] in 2009 and in 2011 [19], presented an authentication mechanism for Hadoop. They used Kerberos authentication control over SSL and three types of tokens include "namely delegation", "block access" and "job" token. The focus of their solution was on communication between the user and HDFS. As the user accesses Name Node through Hadoop's remote procedure call libraries (RPC). RPCs use simple authentication and security layer that uses Kerberos and a delegation token. On the other hand, they secured a streaming socket by a block access control so users by using it could access the Data Nodes. Their approach leads to no-built in security mechanism in Hadoop. Due to the wide usage of Hadoop MapReduce, their work lead to other projects that tried to add different types of security aspects to Hadoop MapReduce.

Hybrid Execution or HybrEx [20] is an execution model for privacy and confidentiality and privacy in cloud computing and the first MapReduce framework designed for the hybrid clouds. HybrEx, has divided into two groups: sensitive and non-sensitive data. It sends the non-sensitive data into public clouds and the sensitive data would be kept in the private cloud. In this model, only four types of MapReduce computation execution are allowed.

Map hybrid, Horizontal partitioning, Vertical partitioning, and Hybrid; In the first one, Mapper is executed at both public and private clouds while the Reducer is executed at a private cloud. In Horizontal partitioning, the data would be encrypted, and the Mapper is executed only at public clouds only while the Reducer is executed at a private cloud.

The third type, vertical partitioning, lets both Mapper and Reducer execute on both public and private clouds, but it will not let data transmission between public and private clouds. And finally, the hybrid type, is the same as the third model with this difference that data transmission among public and private clouds is allowed. The weakness of HybridEx is that it does not deal with the generated keys in Map phases in both public and private clouds.

Sedic [21] provided a solution to solve the key issue in HybridEx. By using an automatic analysis and transformation of the Reduce code, Sedic presented a data movement from the Mapper that executes on a public cloud to the Reducer that executes on a private cloud. The outputs of the Mapper at the public cloud will be aggregated before they are transmitted to the private cloud and by this, the communication between public and private cloud would be decreased.

[22], presented EPiC, a privacy-preserving protocol that evaluates the “Frequency Counting” in MapReduce which is a fundamental operation on datasets. The aim of this protocol is using privacy-preserving counting in MapReduce and by this, the users can safely store their data in public clouds.

The main idea is to transform the pattern search into summation and polynomial evaluations. Through partially homomorphic encryption. Homomorphic encryption is a form of encryption. It allows the computation on cipher-texts to generate an encrypted result that after decryption, matches the result of the operations as if they had been executed on a plain-text [23].

The user encrypts the data and uploads it to the cloud. By this, the cloud provider will not be able to know anything about the stored data apart from trivial information such as data size. This is due to the encryption model which is in a way that no identifiable data value generates a cipher-text. In the next phase, the user specifies a Boolean formula as a searching pattern and generates Mapper and Reducer code to work on the encrypted data. To protect the data from the cloud provider, the computation uses partially homomorphic encryption; eventually, the cloud provider only performs the assigned MapReduce computation and counts the total number of occurrences of an assigned pattern. He will not be able to learn anything about the data, not the pattern and will not be able to know how often it occurs.

The weakness of EPiC is that to increase the efficiency of the execution of the assigned queries, it uses a weaker encryption scheme. It also only EPiC supports the counting operations, which is a limitation for this protocol.

In many of the proposed security solutions, access control had a bold role. Arindam Khaled et al.in 2010 [24] proposed an architecture based on access control and enforcement policy for Resource Description Framework (RDF) [25]. The amount of RDF data on the web is growing at a fast pace, while the access controls for RDF data, scale poorly to large data sets. So, they addressed access control for cloud-resident RDF data. They presented a token-based access control system where users are granted tokens based on the authorization levels and their needs.

The overall procedure in this architecture was that the system administrator, based on the request of users, generates an access token for securely accessing data. The token prevents access to the entire data. They suggested six types of secure data accesses: Predicate data access, subject model-level access, subject and object data access with or without predicates. The challenge was that the RDF format was not suitable for a MapReduce computation, as a result, they proposed a two-layered system. The first layer which was the data processing layer converts RDF to N-Triple format and the second layer which was the query processing layer execute a MapReduce computation. This layer provides outputs regarding an access token. It rewrites a query that satisfies an access token, and then, based on the rewritten query performs a MapReduce job.

In the final step, performs more additional MapReduce jobs to remove sensitive data from outputs according to the access token.

In 2014, Vigiles [26] has been introduced; It is the first system that provides a critical security component for MapReduce, fine-grained access control (FGAC), that challenges for all types of data without modifying the source code of MapReduce mechanism. Vigiles that is a middleware architecture is a layer between untrusted users and MapReduce environment. It uses FGAC for reading operations in MapReduce without modifying computations. However, it only supports read operation and the sensitive data should be removed from reading operation outputs.

Managing Hadoop distributed file system is on system administrator; the administrator also should define the access control filters which are used to remove sensitive data by executing a procedure consists of 3 consecutive phases: decompose, fetch, and action. Access control filters, based on the defined configuration, provide only authorized data to users. In Vigiles, the computation provider (user) writes down the MapReduce code and the code will be executed based on the control policies and it removes the sensitive data from output. However ad-hoc data types and operations such as append and delete are not allowed in this system.

One year after Vigiles, GuardMR [27] solved the issue and allowed ad-hoc data types and provided dynamic access to the record. In GuardMR, the focus is on the automatic generation of dynamic authorized views of input data resources from the high-level OCL specifications.

The access control module performs administrative functions. These functions allow us to add new data types and preprocessing functions after a proper security analysis. After consulting the access control module which leads to an authorized view of data, the systems will reference monitor enforces specified security policies to the underlying MapReduce system. In GuardMR the generated data views are expressive enough to handle the datasets without the need to predefine structures.

3.2 Privacy Preserving Methods

Most of the researches on differential privacy focused on the theoretical properties of the model, and they tried to provide feasibility and infeasibility to the results.

However, Machanavajjhala et al. [28] studied the use of differential privacy in practical applications. They applied a variant of differential privacy to create synthetic datasets. They used U.S. Census Bureau data for this purpose. Their goal was to use them for statistical analysis of commuting patterns in mapping applications.

The challenge was handling datasets with large domains because it is needed that the noise spread throughout the domain even if the data is sparse. The importance of this is because if an outlier appears in the synthetic data, it is more likely that a similar outlier was present in the real data and it is less probable that it was due to random noise.

They used exogenous data and other techniques to help reduce the domain size. Even with domain size reduction, the data was still sparse, eventually, many outliers were created due to the addition of noise to all parts of the reduced domain. So that the distribution of commute distances was reasonable only for the study of not extremely long commutes.

The fundamental question of what it means to preserve the privacy of a computation input has been the subject of much research. Differential privacy is not the first framework that tries to answer this question. Many previous efforts have been done.

The concept of privacy-preserving data mining has been proposed with two broad approaches. The randomization approach which focuses on individual privacy and reveals randomized information about each record while promises to not reveal the original records to anyone.

For example, in [29], they studied the privacy-preserving data mining. They assumed that using a randomizing function, will perturb the user's sensitive data so they can not be estimated with sufficient precision. They tried to answer this question that with a large number of users who do this perturbation, is it still possible to construct sufficiently accurate predictive models? Later [30] proved that their definitions are not sufficient to provide an individual's privacy.

In the secure multi-party computation approach, the goal is that without revealing the individual records in each database, build a data mining model across multiple databases. These privacy concerns can prevent building a centralized warehouse, so in [31], they addressed such problems relate to computing association rules within such a scenario. They assumed homogeneous databases in which all sites have the same schema. However, each site has information on different entities. Their goal was that at the same time which they limit the information shared about each site, they be able to produce association rules that hold globally.

Privacy Integrated Queries (PINQ) [32] is an extensible data analysis platform designed for computations on sensitive data while providing unconditional privacy guarantees for the records of the underlying data sets. This platform ensures differential privacy for the outputs of the computation. It provides a restricted programming language with a small number of trusted primitive data operations in a framework named LINQ.

PINQ uses a request/reply model. This model avoids adding noise to the intermediate results of the computation; instead, it keeps them on a trusted data server provided by a distributed system. PINQ's security is dependent on the security of Microsoft's common language runtime (CLR), the Dryad framework, the Cosmos distributed file system, and the operating system.

The closest work to PINQ is Airavat. Airavat [33] is a system that has implemented differential privacy and provides mandatory access control for data protection. Airavat is the first system that provides roughly a complete solution for secure computation and data privacy in a MapReduce system. Airavat uses a mandatory control system to ensure that untrusted mappers do not leak data outside the cluster. It does not allow direct access of mappers to the data and the network. It uses Security-Enhanced Linux (SELinux) to implement access control. Unlike PINQ which provides language level guarantees, Airavat's privacy enforcement mechanisms provide end-to-end guarantees.

This thesis has borrowed the same philosophy of Airavat however the technical part of the implementation is completely different as it was not possible, we go through the same implementation of Airavat due to fundamental changes that have occurred to the software that Airavat integrates into. We also added new features to the implementation such as combiner and we avoided making changes in Hadoop main source code which this latter not only had a positive effect on performance due to low overhead but also added flexibility to Parmanix as with each Hadoop update, we do not need to change Parmanix main code.

3.3 Additional Discussions

In this section, we discuss RQ1 on why Hadoop MapReduce needs a privacy solution and why noise-based privacy solutions can be a practical solution to achieve this goal.

3.3.1 Need for a Privacy Solution in MapReduce

The question that we try to answer is why we need a privacy solution in data analytics and why we chose Hadoop MapReduce platform for this purpose?

Hadoop MapReduce is an open-source venture. MapReduce is extensively used around the world as an efficient distributed computation tool for a large class of problems such as search, clustering, analysis of social networks, log analysis, etc. We chose the Hadoop platform as it has been successfully used by many companies such as Amazon, Facebook, Yahoo, and even the New York Times for running their applications on clusters. Hadoop has several tools which its main one is MapReduce. In fact, Hadoop has been designed for MapReduce.

Hadoop has various modules that each of them has been separately developed over time in order to add different types of functionalities to Hadoop. However, the security of Hadoop was never at the center of attention for development until recently. So, while Hadoop has been increasingly used, the weakness of the security mechanism has become one of the main challenges that obstruct its development. Eventually, Hadoop lacks a consistent security model and there are a lot of security threats that can hamper the operation of MapReduce and other components of the Hadoop framework components.

According to the reports [34], Hadoop is easily identifiable to hackers all around the world. They do this by simply sniffing to open instances.

Being open-source and accessed by so many users all around the world and having security weaknesses in one hand and the ascending growth of data analytics, on the other hand, made us choose Hadoop MapReduce platform and put the focus on improving its privacy gaps.

Parmanix is a platform for privacy-preserving data analysis. To provide the privacy that has been promised, it uses the Differential Privacy algorithm and integrates it into Hadoop MapReduce, in order to aggregate features from input datasets without leaking any information about any specific data item.

We mark a computation on a set of inputs as differentially private if, for any possible input, the computation output does not depend on the presence of that input in the data set; or as Cynthia Dwork has defined it: "The outcome of any analysis is essentially equally likely, independent of whether any individuals join or refrains from joining the dataset." [15]

Note that in Parmanix, we have targeted not only privacy but the security of Hadoop MapReduce. This means that our perception is that the intruder tries to get some information about an individual, while we block some of the ways he can steal data through data leakage (unsecure Reducers), we take the certainty of the intruder about existence or non-existence of an individual's data in the output of the MapReduce and by this protect the privacy of the individual.

Experiment

Assume we have a dataset containing information of employees in a company. We assume this company has 50 employees. 30 of them work in department A and 20 of them work in the department B. Table 3.1 shows 6 records of this dataset. As can be seen, this dataset has 6 attributes. Among them is SIN number which is one of the riskiest attributes that its leakage will cause so many security problems for the person.

Table 3.1 Sample Records of the Dataset

Employee Number	Employee Name	Department	SIN Number	Salary Per Year($)	Gender
12356	David	A	262-353-938	90,000	Male
12990	Charlotte	A	743-234-278	100,000	Female
21723	Jessica	B	346-837-338	100,000	Female
21009	Sophia	B	983-363-122	100,000	Female
12005	Charles	A	343-384-226	90,000	Male
12843	Olivia	A	645-999-393	100,000	Female

Through this dataset, we can query the total income of employees that work in Department A. David is one of the employees of this company that has left the company in the past month. We intend to retrieve information about his salary. The background knowledge that we have in this scenario is that David has left the company in the past month and that the IT department has access to all datasets related to the company employees.

In the original Hadoop MapReduce, it is possible that simply query for that specific person and get the information we seek. However, we assume the intruder does not have direct access to the dataset itself and he can gain the information he wants only through indirect queries he asks from the IT department. Note that the IT department does not share personal information of employees.

He can ask the IT department for a total income of employees of department A of the company in 2 different months; The month that David was working and the month that David left the company. Table 3.2 shows the results of the queries.

Table 3.2 Total Income of Employees in Department A

Total Income Before David leaves the Company	2,500,000$
Total Income After leaves the Company	2,410,000$

The intruder can simply compare the output results and find out David's salary was 90,000$ per year. As can be seen, MapReduce simply exposes the individual's data and there is not any mechanism on the platform itself to protect the data. The output results with and without that person's data in the dataset are different and this reveals information about individuals.

MapReduce is dependent on dataset security itself. This means if the dataset has not been anonymized, the platform is not providing any privacy solution. Note that in this simple example we ignored that the intruder can straightly query the data record and we put our assumption on the disability of the intruder to straight access to the dataset and that he indirectly and by the help of his background Knowledge - that David left the company last month - and the intermediaries - which was IT Department- retrieved the data he wanted.

Parmanix provides privacy at the platform level. So independent from the dataset, whether it was anonymized or not, Parmanix guarantees privacy for each of the participants in that dataset.

3.3.2 Differential Privacy Preponderance Over Other Privacy Solutions

In this section, we try to answer why we chose Differential Privacy as our solution? Were not other solutions enough for providing privacy?

Let's first review what Differential Privacy does and later make a comparison between Differential Privacy and other existing solutions. The definition of Differential Privacy was first proposed in Cynthia Dwork's ICALP paper [15]. Since then so many contributions in terms of both theoretical analysis and practical instantiations have been taken place.

Differential Privacy addresses the case when the data provider wants to release some statistics and does not want to reveal any information about a particular value itself. Differential privacy makes it impossible to identify the individual's records in that dataset.

It does this by injecting noise into a dataset, or into the output of a machine learning model. The positive point about this is that it does it without any significant negative effects on the final output. To get this satisfying output, the differential private platform calibrates the noise level to the sensitivity of the algorithm. The final result is a differentially private dataset or model that cannot be reverse-engineered by an intruder.

3.3.2.1 Data Anonymization

To answer the research question we stated, we review one of the most popular privacy solutions that currently is widely used, “Data Anonymization”. It refers to techniques that are designed to prevent the identification of a particular individual from the dataset that contains this information.

Same as differential privacy, the goal of data anonymization is to protect the individual’s privacy and it does it through different methods such as encryption, pseudonymization, hashing, etc.

There are some special kinds of attributes in a dataset: Identifiers and Quasi-identifiers. Identifiers allow us to easily identify a person, such as a SIN number or a name. Quasi-identifiers are a broader term that includes identifiers and other attributes that may appear in other public databases and may allow identifying the person linked to the data. An example of this can be gender, postal code, or race.

Data anonymization is the process of either encrypting or removing identifiers which are the information that is personally identifiable from data sets, so that the people whom the data describe remain anonymous. It does it through different methods; Such as converting clear text data into a nonhuman readable and irreversible and by this, it reduces the risk of unintended disclosure, so this allows the transfer of information across a boundary, such as between different institutes and companies. Or by removing all identifiers such as Name, Social Insurance Number, Address, etc. that are unique to a person. [35] has described some of the anonymization algorithms.

But why anonymization is not enough? Why Differential Privacy can be a better solution?

Even with removing the explicit identifiers, the data is still vulnerable as the intruder uses the quasi-identifiers to link the dataset with another dataset that can be available publicly and still contains explicit identifiers. This call “linkage attack”. There are several examples of anonymization approach failure that costs the individuals, exposer of their very sensitive data.

Failure Example 1

In the mid-1990s, in Massachusetts, USA, the Group Insurance Commission (GIC) purchased health insurance for state employees. This information included patient-specific data on 135,000 employees and their families; They decided to release some of this data for research purposes. To answer the concerns regarding personal information that existed in those datasets, the governor of Massachusetts William Weld claimed the privacy of individuals would be protected as explicit identifiers such as name and address, had been removed.

To prove these claims are incorrect, an MIT student, Latanya Sweeney, bought the voter registration list for Cambridge, Massachusetts for just 20 dollars. She used that data to link it to the GIC dataset by using birth date, gender and ZIP code. She retrieved health records of governor Weld's health and delivered it to his office.

She stated that according to the Cambridge Voter list she found out there were six people that had the same birth date as Weld which three of them were men; however, he was the only one in his 5-digit ZIP code. [36] [37]

Figure 3.1 shows the common quasi-identifiers of GIC and Voter datasets.

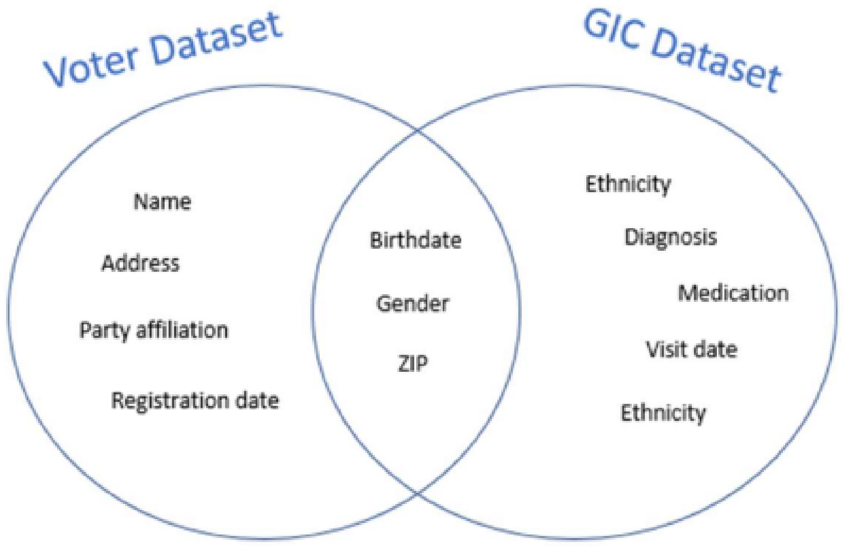


Figure 3.1 Sweeny Linkage Attack to GIC Dataset by Using Voter Dataset

3.3.2.2 K-Anonymity

A popular approach for data anonymization is K-anonymity [38]. k-anonymity transforms an original dataset containing an individual's information to make it difficult for an intruder to determine the identity of those individuals in the dataset. If every combination of values for demographic columns in a dataset appears at least for k different records, we say that the dataset is k-anonymized.

A k-anonymized dataset has the property that each record is similar to at least another k-1 other records on the potentially identifying variables. For example, if k = 5 and the potentially identifying variables are “Postal Code” and “Birth Date”, then a k-anonymized dataset has minimum 5 records for each value combination of “Postal Code” and “Birth Date” [39]. Any record in a k-anonymized data set has a maximum re-identification probability of $\frac{1}{k}$ [40].

Higher values of k imply a lower probability of re-identification; It also means more distortion to the data which leads to greater information loss due to k-anonymization. Note that excessive anonymization will make the data less useful for analytics as some analysis becomes impossible or produces incorrect results [41].

In fact, Sweeney proposed k-anonymity as a new privacy model after her re-identification of the GIC dataset to ensures that it is not possible to distinguish each individual’s quasi-identifiers from at least $(k - 1)$ others.

While it seems that k-anonymity will solve the issue, the flaws have been showed up. Assume that a group of k records with the same quasi-identifiers have the same sensitive attribute value; In this case, once again it is possible to re-identify the individual records. There are many examples of the failure of k-anonymization approach in protecting individual’s data.

Failure Example 2

The biggest example of k-anonymization vulnerability is “Netflix prize competition”.

This competition released movie ratings from subscribers of Netflix which is the world’s largest online movie rental company. Netflix released a training dataset comprising 100 million ratings from nearly 500,000 subscribers on 18,000 movies in a seven-year time-frame to aid researchers for training purposes. Rating data included the rating itself (from 1 to 5stars), the date of the rating and the movie. To protect subscribers’ privacy, all personal information that could lead to their identification was removed. Although the data was carefully anonymized, however, Narayanan and Shmatikov [42] later “de-anonymized” some of the private records.

They had access to auxiliary data in the form of a public, un-anonymized dataset from the IMDB website which is a movie-related website and contained similar ratings. In fact, they used the IMDB dataset as their source of background knowledge, applied their de-anonymization methodology to the dataset and identified the subscribers' record in the dataset. It even included uncovering their apparent political preferences and other potentially sensitive information. [43]

Failure Example 3

Culnane et al. [44] have shown re-identification of patients in an anonymized health dataset published by Australia's health department. In 2016, pursuing the Australian government's policy of open government data, the federal Department of Health published the anonymized longitudinal medical billing records of nearly 2.9 million people. They showed that without even de-anonymizing the dataset and only by linking the parts of the record that exists in the dataset to the known information about the individual, they can reveal the identity of each person such as the birth year of mothers and their children.

Failure Example 4

The Freedom of Information Law (FOIL) allows public access to the records of governmental agencies in New York state. In March 2014, the New York City Taxi and Limousine Commission (TLC) tweeted a graph that made blogger Chris Whong [45] use FOIL to obtain the information of all New York City taxi trips taken in 2013. It happened through reverse-engineering the anonymized data to find out the driver and the taxi for each trip. The data had been anonymized by hashing from an anonymous data set of individual trips in the city.

The original data contained information regarding fares and trip including total fare, surcharge, toll, date, GPS coordinates, distance traveled, etc. Fare and trip data were in two tables that linkable by a hashed medallion and hack license identifier. Although the fares linked to their corresponding trip, but the identity of the taxi driver and the passenger were unknown till three months after receiving the exposed data, another contributor noticed that the "hack license" and "medallion numbers" identifiers had been created using the md5 hash function.

He calculated the md5 hash for all possible "hack license" and "medallion numbers" and eventually the information of 22 million people re-identified. After this, TLC removed the "hack license" and "medallion numbers" from the datasets of 2014.

3.3.2.3 General Inference

Now we get back to our original question. Why these solutions are not enough?

The answer is that these solutions ignore the fact that the intruder can have auxiliary information and by using them can reach to individual's data. This auxiliary information can be his personal knowledge or be the information that he has retrieved from many public datasets that exist on the internet. Eventually, he can simply use these datasets and by querying and comparing the output results, find the information he seeks. Such as what happened in the Netflix example.

The arbitrary information can be (or become) available to the intruder and as a result such errors are inevitable to happen as they are hard to prevent. Ignoring the background knowledge is the biggest weakness of these solutions. While in differential Privacy, even though the intruder can have any background knowledge, the certainty has been taken from them and they can never access to individual's data records.

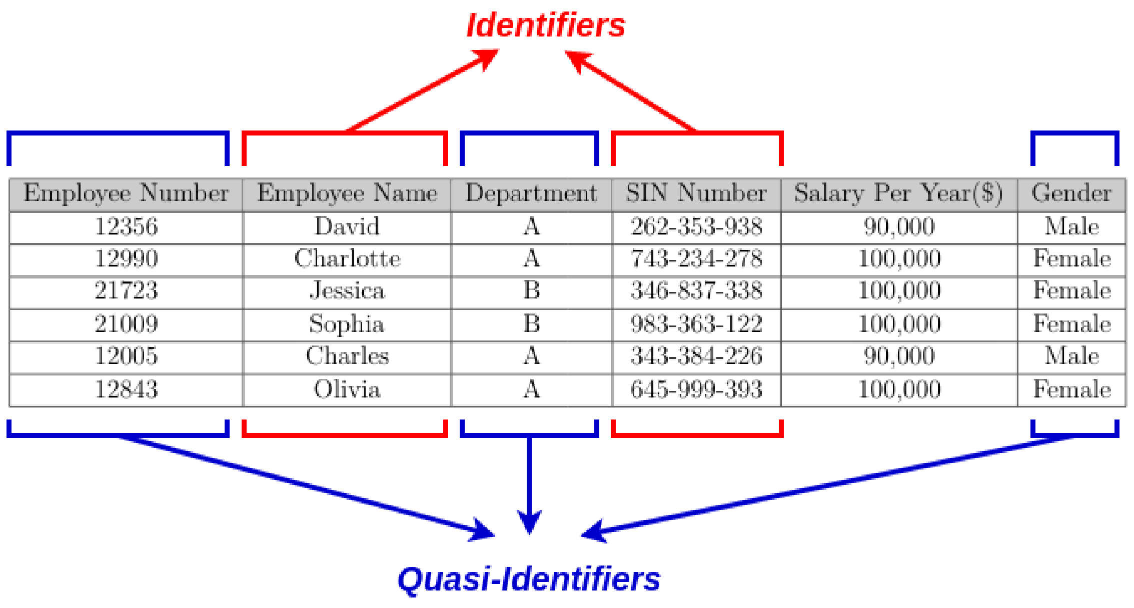
In the next part of this section, with a theoretical experiment, we show the function of k-anonymity and Differential Privacy in protecting individual's data.

Experiment 1

Same as the scenario in section 3.3.1 , the intruder wants to discover David's salary who was one of the employees of the company and he has left the company in the previous month. We assume the previous dataset has been k-anonymized. The $k = 2$ and it means there should be at least two or more than two values in an attribute of the data record that have similar values. In other words, there should not be any singular value in there.

To show how 2-anonymity applies to dataset records, consider the 6 records in table 3.1. We try to use suppression method – which replaces individual attributes with a * – on these 6. Note that these are just 6 records out of a 50-record dataset, so we try just to show how the concept of k-anonymity works by using these 6 records as a sample.

First, the identifiers should be removed. Table 3.3 shows the identifiers and quasi-identifiers. We omit the "Name" and "SIN Number" columns and in the next step apply 2-anonymity to the other quasi-identifiers.


Identifiers
Quasi-Identifiers

Employee Number	Employee Name	Department	SIN Number	Salary Per Year($)	Gender
12356	David	A	262-353-938	90,000	Male
12990	Charlotte	A	743-234-278	100,000	Female
21723	Jessica	B	346-837-338	100,000	Female
21009	Sophia	B	983-363-122	100,000	Female
12005	Charles	A	343-384-226	90,000	Male
12843	Olivia	A	645-999-393	100,000	Female

Table 3.3 Identifiers and Quasi-identifiers of The Dataset

The “Employee Number” column, has unique values. We remove the last 3 digits of employee numbers. Now we have at least 2 records that are similar in that column. Table 3.4 shows the column after suppression.

Table 3.4 After Applying 2-Anonymity To The “Employee Number”

Employee Number	Department	Salary Per Year($)	Gender
12***	A	90,000	Male
12***	A	100,000	Female
21***	B	100,000	Female
21***	B	100,000	Female
12***	A	90,000	Male
12***	A	100,000	Female

Now we look at the Department column. It does not need any changes as each Department, has been repeated two or more than two times. The next column which is “Salary”, has the same situation. In the last column, there is no singularity as well and eventually, no change is needed.

We have a dataset that there is not any unique row. There are at least 2 or more than 2 rows with similar values. In table 3.5 rows with the same color are the same as each other after applying 1-anonymity.

If through Hadoop, computation provider requests the total salary of department A employ-

Table 3.5 Anonymised dataset; There are at least 2 or more items that are the same

Employee Number	Department	Salary Per Year($)	Gender
12***	A	90,000	Male
12***	A	100,000	Female
21***	B	100,000	Female
21***	B	100,000	Female
12***	A	90,000	Male
12***	A	100,000	Female

ees, for two different times, one for when David was still working in the company and one for when he left the company, the output would be still the same as before. As can be seen, anonymization can not guarantee the protection of individual's data as the intruder simply by knowing that David is not working in the company and a comparison between results can find out David's salary.

Experiment 2

This time we assume the same scenario but in Parmanix platform. We write the Mapper and query the total salary of the employees of department A and use the secure Reducer Sum for this purpose and define the privacy parameters. The details related to how we write Mapper and Reduce and what are the privacy parameters and how we define those parameters will be explained in Chapter 4. We intend to illustrate how Parmanix overcomes the flaw of k-anonymity.

We query the total salary of Department A employees and for 2 different dates, one for the month when David was working at the company and one query for the time that he is not an employee anymore.

Parmanix is not supposed to give out the exact answer. According to the parameters that the computation provider declares for it, the output will be a noisy answer that hides the actual answer. Table 3.6 shows us the output.

Table 3.6 Total Income of Employees in Department A (Parmanix)

Total Income Before David leaves the Company	2,490,000$
Total Income After leaves the Company	2,460,000$

These answers are not the actual answer and the intruder can not gain any information from it. Possibly he assumes the salary of David is 30,000$!

So Parmanix guarantees that while you can use the output for data analytic purposes, but you can not gain any info from individuals in that dataset.

CHAPTER 4 METHODOLOGY AND DESIGN

4.1 Context and Problem

Privacy ensures that sensitive data is not exposed to untrusted users such as adversaries, data providers, computation providers, etc. Generally, the data providers are interested in allowing computations on the data, however, it is required to preserve breach of sensitive data.

Parmanix is a module that uses Differential Privacy as a noise-based privacy solution in Hadoop MapReduce which is a data analytic platform. The goal of this empirical study is to protect an individual's information in a dataset while doing the data analytic. Implementation of Parmanix has been inspired by previous research work, Airavat [33]; Airavat was using this algorithm in MapReduce, however, they had a different approach in their implementation as it was combined with an access control mechanism and needed to change Hadoop Source code and JVM source code through patches. In contrast, we designed Parmanix to be added to Hadoop like a module without any modification in the source code. Besides, unlike Airavat that had the limitation to use a combiner, we can use combiner and this has an impact on the utility.

4.2 Study Definition and Design

Our empirical study aims at implementing the Differential Privacy algorithm through Hadoop MapReduce and assessing the effectiveness and utility of our proposed approach in different scenarios. In order to keep the performance acceptable, Parmanix gets added to the platform like a module without making any changes in Hadoop source code.

In the following, we introduce the answer to research questions 1 and 2 which have been mentioned in the first chapter; This includes a description of Parmanix architecture and how this module affects privacy and accuracy.

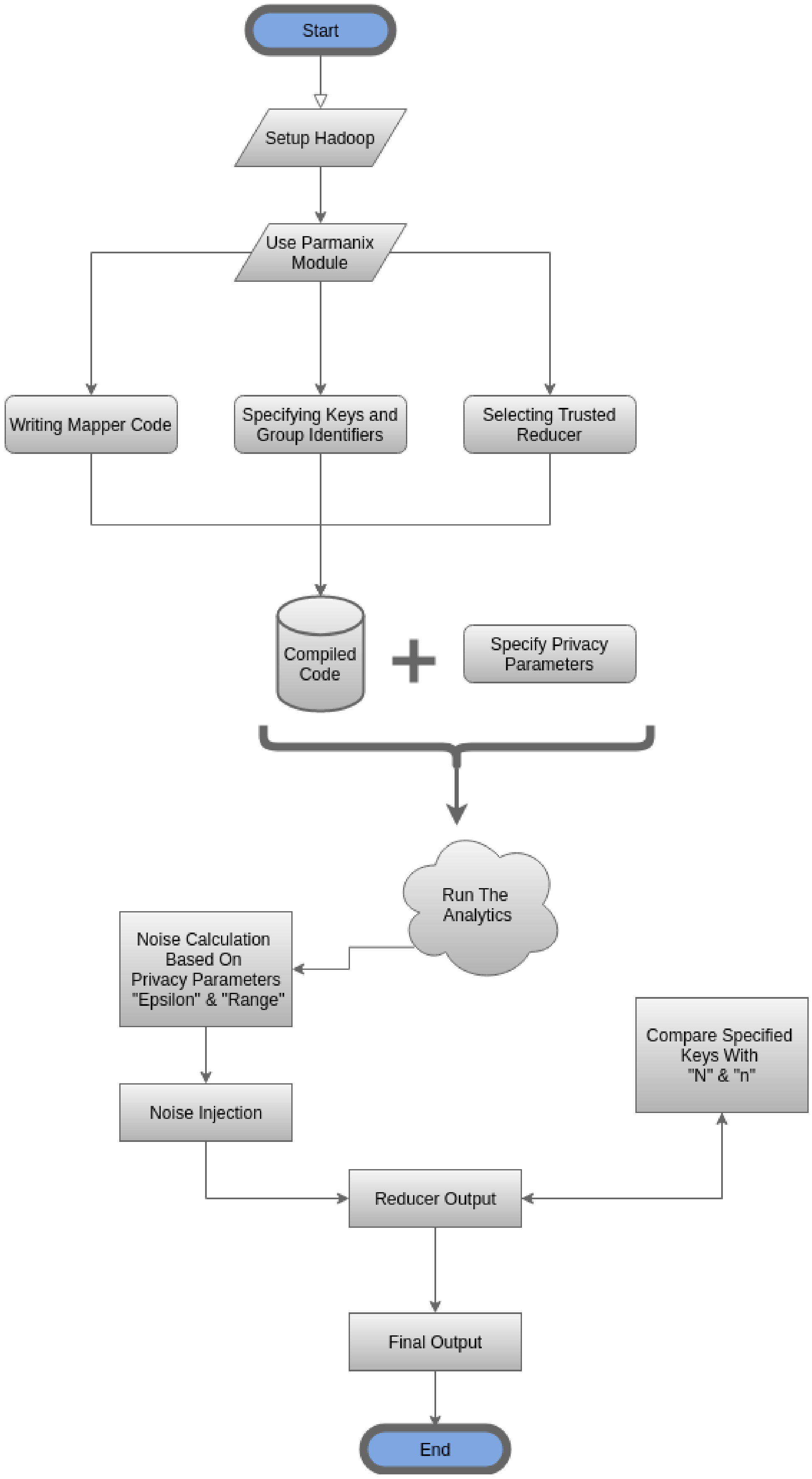

Start
Setup Hadoop
Use Parmanix Module
Writing Mapper Code
Specifying Keys and Group Identifiers
Selecting Trusted Reducer
Compiled Code
Specify Privacy Parameters
Run The Analytics
Noise Calculation Based On Privacy Parameters "Epsilon" & "Range"
Compare Specified Keys With "N" & "n"
Noise Injection
Reducer Output
Final Output
End

Figure 4.1 Parmanix Module Model

4.3 Parmanix Programming Model

4.3.1 Untrusted Mappers, Trusted Reducers

In the original Hadoop MapReduce, the computation provider writes his own Mapper and Reducer code and runs it as a jar file through Hadoop. In Parmanix this procedure has been changed. Figure 4.1 shows this module model. In our approach, by inspiration of Airavat, the user writes the Mapper code the same as before. We do not impose any restriction in this part for the user and eventually, we call it the unsecure Mapper. However for Reducers, the computation provider will not write his own code, instead, he uses the Reducers that we have already defined. The Reducers are responsible for the privacy enforcement and that is why they have to be trusted and secure.

4.3.1.1 Writing Mappers

As figure 4.1 shows after setting up the Hadoop platform, in order to use Parmanix module user has 3 tasks to do.

The first task is to write his Mapper code. In our privacy preservation module, the Mapper is not allowed to save any <key, value> pairs. This means they cannot use the already produced <key, value> for a specific record and use it when computing another <key, value> pair. Mappers cannot merge the information from other records. All the keys they produce should belong to the same record. If not, estimated sensitivity would be lower than the actual one which risks the privacy. This means that all invocations of a Mapper in the computation are independent. To implement this Mapper independence, we restricted Mapper to read any record during the Mapper initialization.

In the implementation of Parmanix, we have defined 9 main classes. We devoted two classes “ParmaMapper” and “IdentityMapper” to the computation provider (Parmanix user) so he can write his Mapper code. We call it the untrusted Mapper as we do not do any privacy checking or imposing any restriction on this code. Figure 4.2 shows the classes of Parmanix.

There are 6 other classes that the user does not make any changes in them. "MapperWrapper" and "MapperInterface" are the classes that receive the input data and execute the untrusted Mapper that the user has written.

"MapReduceConfigured" is the class that we have defined the privacy parameters (We will describe these parameters in the next section). "ParmaCombiner" is where we have defined the combiner and "ReducerWrapper" is the class that we have defined the differential privacy algorithm. We have defined all of our trusted Reducers in the "TrustedReducer" class.

4.3.1.2 Specifying The Keys

Privacy Parameter N: The Mapper code that computation provider writes might be buggy or contains malicious codes that try to leak information in the output, but using the secure, predefined Reducers will block this way of data leakage while letting the data provider to not need to audit the inspect computation provider source code. In our approach we aim to block 2 possible ways that can lead to information leakage through the untrusted Mapper code:

One way can be through either key or value. We know the output of Mapper is <key, value> pair and computation can have more than one <key, value> pair. We can never know if a key is encoding sensitive data or not. This can be a method for a malicious Mapper to steal data because observing a specific key in the output can be an indicator of an individual's information. To block this type of data leakage, the computation provider must submit ***the list of key(s)*** and the ***privacy parameter "N"*** before the start of computation. Privacy parameter N indicates the number of output keys and is used when the key numbers are less or more than the key numbers that have been specified by the computation provider.

In such a case that the number of output keys does not match with declared keys, Parmanix will add or remove outputs to make it the exact number of declared key numbers. In fact, N declares that when the computation has been done, how many keys we must see as an output. Because the Mapper is untrusted, and a malicious code that has been written in it can try to leak information and as a result, we see more (or less) results than what we queried.

The adversary can get some information about individuals simply by adding or removing different keys and comparing their outputs and we want to block this way of data leakage. By declaring the list of keys and the privacy parameter N, even if none of the Mappers produce the submitted key, Parmanix still will return a value.

The second way for information leakage is through the output values of different keys. The adversary can emit a certain combination of values related to different keys through malicious Mapper. In Parmanix platform, the probability of producing a combination of output values related to different keys would be the same, with or without a specific input item.

Eventually, no data will be leaked through any combination.

Privacy Parameter n: One of the concepts that we have implemented in Parmanix and it is inspired by Airavat, is “Privacy Groups”. This concept targets the datasets that are different in more than one item and in fact they are different in a group of records. In such a case, Privacy groups let us extend privacy to not only items but groups. Privacy groups, in fact, are a collection of records that together can be present or absent in the dataset. In order to implement them, we used the composability of Differential Privacy. Composability simply says that: the effect of **n** input items on the output would be maximum n times the effect of one item.

To depict this concept, assume a dataset of a company’s sold items. A customer can buy different items.

Table 4.1 Customer’s purchase

Customer Name	Product	Date
Bob	T-shirt	2018/09/01
Bob	Pants	2018/09/01

Table 4.1 shows the customer’s two purchases. Instead of counting each row of this table as a record, we count them as one group. This group is all of Bob’s purchases and this is the data provider who should provide the ***group identifier*** and the ***privacy parameter n*** based on their dataset so the computation provider can use it for analytic purposes. “n” which is the maximum number of privacy groups’ keys do the same job as N but in the privacy group level.

The second task that the user must do is to specify the keys. The computation provider must declare what are the expected keys and also defines the group identifiers. We previously explained that this is to block data leakage through Keys and declare the two privacy parameters, N and n which are respectively the number of expected outputs and the maximum number of privacy groups’ keys. He will use the “MapperWrapper” class for this purpose.

Note that MapReduce operations might execute on different nodes. For the inputs that go to different nodes, “Key” and “Privacy Group” will be the same. The group identifiers that the user has defined in the class later will be attached to the <key, value>. By this, Parmanix will be able to track the information through intermediate levels to the output level. We also assume the declared rang by computation provider as the group level.

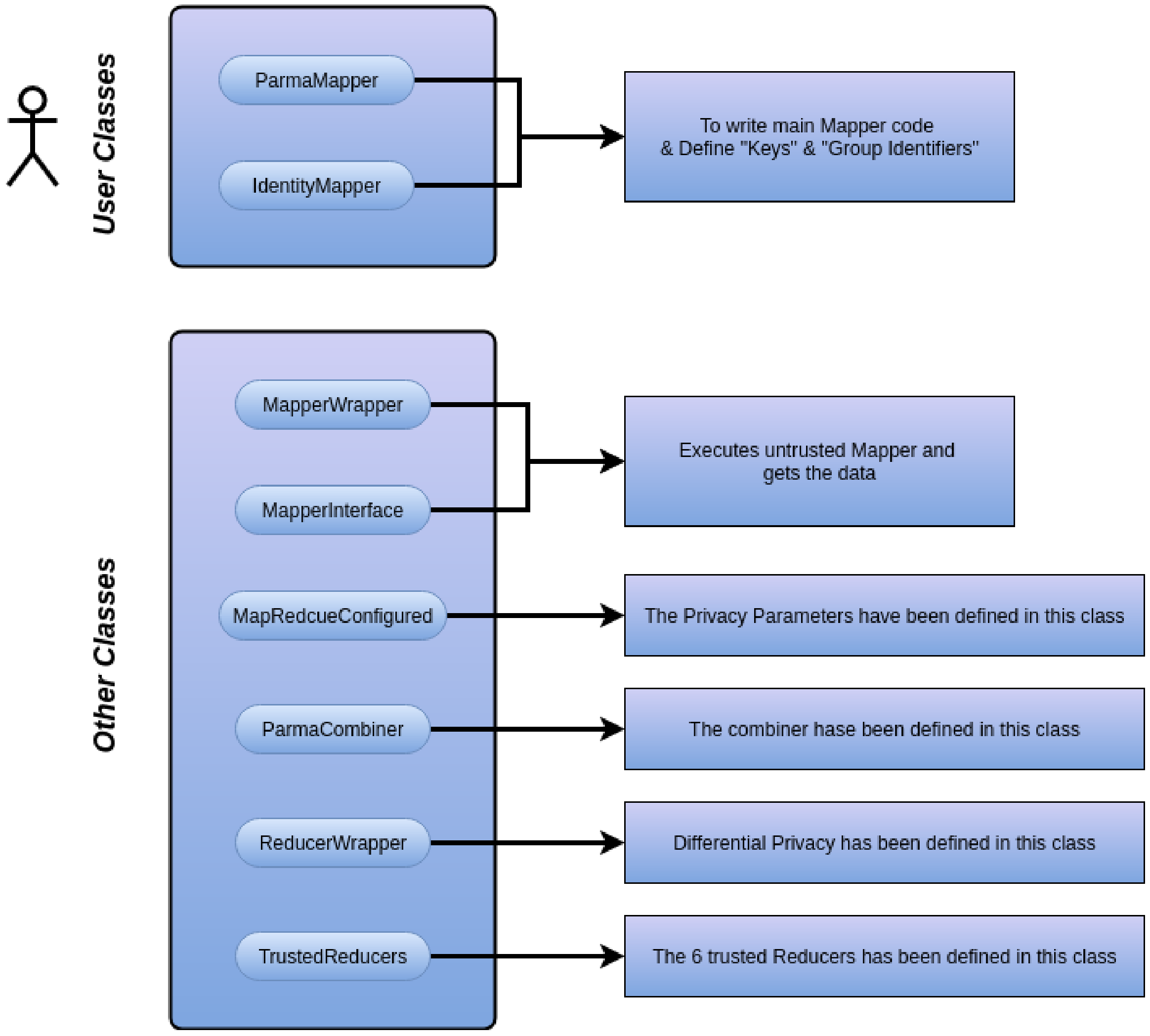
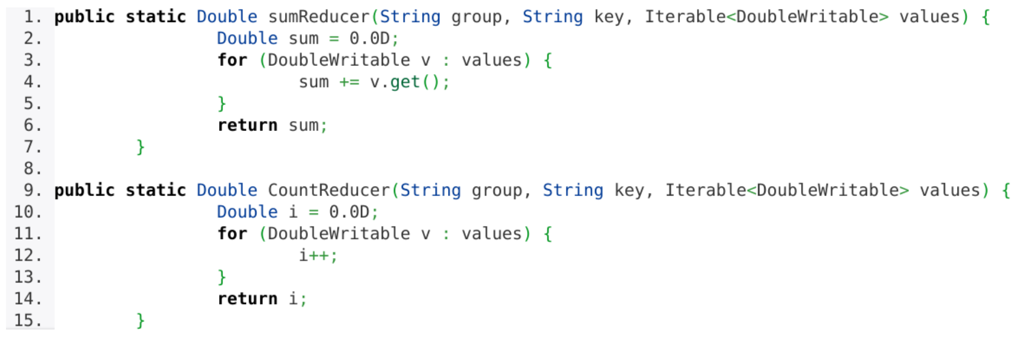

User Classes
ParmaMapper
IdentityMapper
To write main Mapper code & Define "Keys" & "Group Identifiers"
Other Classes
MapperWrapper
MapperInterface
Executes untrusted Mapper and gets the data
MapRedcueConfigured
The Privacy Parameters have been defined in this class
ParmaCombiner
The combiner hase been defined in this class
ReducerWrapper
Differential Privacy has been defined in this class
TrustedReducers
The 6 trusted Reducers has been defined in this class

Figure 4.2 Parmanix Classes

4.3.1.3 Secure Reducers

The third task that the user must do is to decide the reducer he will use. We have defined 6 trusted Reducers (Table 4.2). The user will not write any Reducer code by himself. He will use the Reducer name later in the command line to execute the computation. Figure 4.3 shows the sample code of two Reducers, "Sum" and "Count"



```
public static Double sumReducer(String group, String key, Iterable<DoubleWritable> values) {
                Double sum = 0.0D;
                for (DoubleWritable v : values) {
                        sum += v.get();
                }
                return sum;
        }

public static Double CountReducer(String group, String key, Iterable<DoubleWritable> values) {
                Double i = 0.0D;
                for (DoubleWritable v : values) {
                        i++;
                }
                return i;
        }
```

Figure 4.3 Trusted Reducers: Sum and Count

We also have defined combiner in the "ParmaCombiner" class. Combiner is by default enabled. The Mapper outputs go to the combiner first to be processed and in the next step, combiner output will be transferred to the selected Reducer.

Table 4.2 Parmanix Trusted Reducers

Trusted Reducer	Description
Sum	To sum the values
Count	To count the values
min	To calculate the minimum value
Max	To calculate the Maximum value
Mean	To calculate the average
Median	To calculate the "middle" value in the list

Algorithm 1 shows the simplified MapReduce pseudo-code in the Parmanix module. As can be seen, the user must select from one of the 6 Reducers or the computation can not continue.

Algorithm 1 Simplified MapReduce Pseudo-code

```
Input
    List of Keys
    List of Group Identifiers
    Dataset D
    MinRange, MaxRange
    Epsilon
Output
    Noisy Value L

procedure MAPPER
    GetOutputKey ($K_1, ...K_n$)
    GetInputKey (GroupList)
    Computation Provider writes Mapper
    if $MaxRange > MinRange$ then return "$intermediate < key, value >$"
    else
        "MinRange needs to be smaller than MaxRange"

    procedure REDUCE
        classTrustedReducer
        "User must select the name of trusted Reducer in the execution command line"
        if user SELECTED Sum then
            Method SumReducer
        else if user SELECTED Count then
            Method CountReducer
        else if user SELECTED Max then
            Method MaxReducer
        else if user SELECTED Min then
            Method MinReducer
        else if user SELECTED Mean then
            Method MeanReducer
        else if user SELECTED Median then
            Method MedianReducer
        else
            "Reducer is not valid. Please choose a reducer from the list: sum, min, max,
mean, median, count"
        Return Reducer-Output


        procedure DIFFERENTIAL-PRIVACY ENFORCEMENT
            Noise calculation based on Epsilon and Min/MaxRange
return Reducer-Output + Laplacian Noise
```

4.3.2 Differential Privacy Enforcement

The first part of our methodology was writing the Mapper code, (includes declaring the keys and group identifiers), selecting the predefined Reducer and specifying the privacy parameters "N" and "n".

After writing the code, compiling it and producing the jar file, the next step would be the specification of Differential Privacy parameters so Parmanix is able to produce the appropriate amount of noise. We have used a Laplacian Differential Privacy mechanism that adds Laplacian-distributed noise to a function; we can define it as follows:

$$f(x) + (Lap(\Delta f/\epsilon)) \tag{4.1}$$

Algorithm 2 shows the pseudo-code of our algorithm. The sensitivity of f(x) function denoted Δf, is a measure of how revealing the function might be and adds Laplace noise with scale Δf/ϵ to preserves ϵ-differential privacy.

Algorithm 2 Differential Privacy in Parmanix

```
Input
    $F(X) \leftarrow$ Reducer-Output
    $\epsilon \leftarrow$ Privacy Parameter
    MinRange
    MaxRange
Output
    $F(X)+$ Laplacian Noise

procedure DIFFERENTIAL-PRIVACY:
    //Calculate the sensitivity:
    $\Delta F = |MaxRange - MinRange|$

    //Calculate the proper amount of noise:
    $L = Lap(\Delta f/\epsilon)$
    Inject $L$ into $F(X)$
    Return $F(X)+$ Laplacian Noise
```

Based on our Differential Privacy code written in “ReducerWrapper” class, Parmanix needs 2 privacy parameters, “Epsilon (ϵ)” and “Sensitivity” in order to calculate the proper amount of noise to be added to the result and to guarantee bounds on how much information might be revealed about someone who is participating in a database.

Privacy Parameter “Epsilon”: As described in chapter 2 Epsilon (ϵ) is one of the main parameters in Differential Privacy and eventually in Parmanix. In computing the sensitive data, the statistics that usually get released as the output, pose some privacy risks. The parameter is used to quantify this privacy because it is a metric of privacy loss at a differentially change in data. Note that epsilon is a relative measure of privacy and not an absolute measure. As epsilon gets smaller the privacy gets stronger and vise-versa. We have shown ϵ effects on the computation in the evaluation chapter. The user must specify this parameter.

Privacy Parameter “Sensitivity”: Another impactful parameter in DP noise calculation is “Sensitivity”. It is the amount of changes that occur to the output when one of the input records is omitted. In Parmanix the Reducers add exponentially distributed noise to enforce the Differential Privacy. Sensitivity is the factor that determines how much noise is sufficient to be added to the output. Sensitivity mathematical definition has been given in chapter 2. Based on our definition of sensitivity, the sensitivity of the “Count” function is 1. Because the count can be changed maximum 1 if an item be added or removed to the dataset

$$\max(|M_{min}|, |M_{Max}|) = 1, \tag{4.2}$$

or sensitivity of the “Sum” function depends on the range. For example, in a bound range of integers from 0 to 100, the sensitivity is 100. Because if 100 be added or removed, the output will be affected by 100 units.

In Parmanix to calculate the sensitivity, it is required that the data provider specifies the **Range**; The computation provider must declare what would be the Minimum and Maximum of the range. This will be used to calculate the sensitivity as we can derive an estimation of sensitivity as the sensitivity will be computed based on the declared range.

The range declaration is completely based on the data values in a dataset and the query. So data provider must evaluate their dataset and define what they need to extract from it and based on these he decides what would be the minimum and maximum of the range.

For the "Count" function, the minimum and maximum of the range would be between 0 and 1. For the "Sum" function it would be between 0 and the biggest number. By declaring the range, Δf which is the sensitivity will be calculated and the noise will be computed.

$noise \sim Lap(\Delta f/\epsilon)$

As the computation runs, the computed noise will be enforced to the Reducer output.

We can say the higher the sensitivity of a function the more data leaks about the presence or absence of a specific item in the input dataset. Even one of the indicators of a malicious function is its high sensitivity because it will leak more information by running the function and this can be suspicious whether this is a try to steal data or just the nature of that function.

Figure 4.4 shows the general concept of our module.

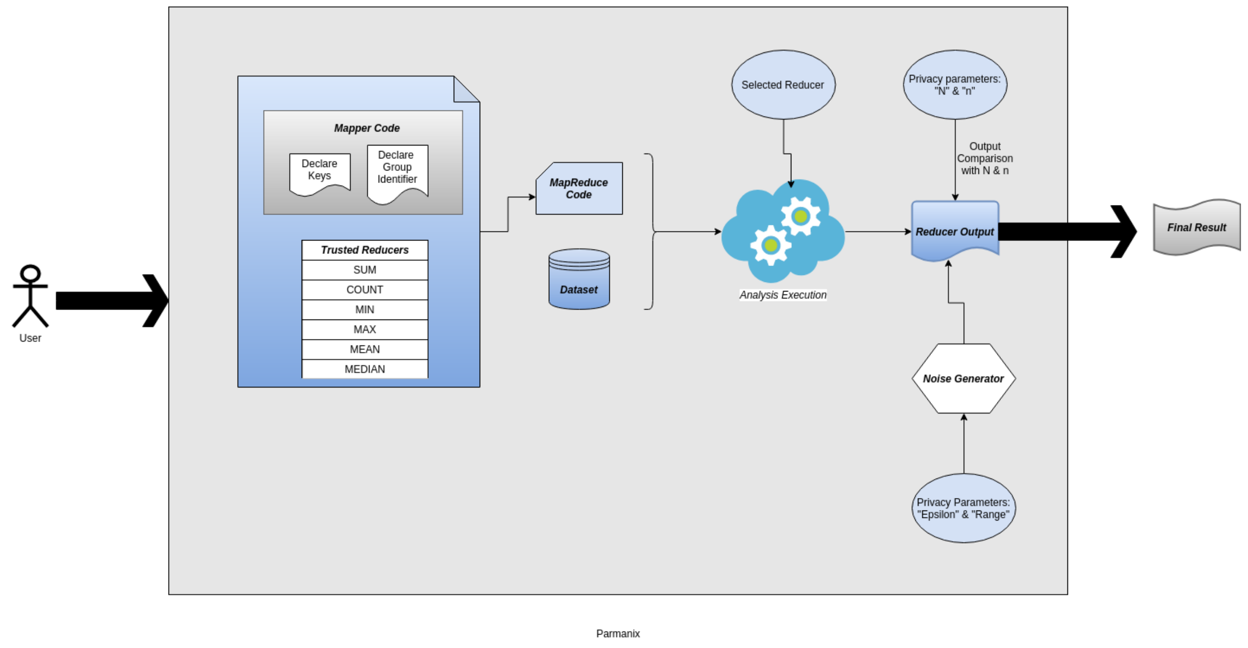


Figure 4.4 Conceptual Architecture of Our proposed module

The calculated noise will be added to the output of Reducer. There is no specific instruction that how data provider and computation provider should calculate these parameters. It is up to them that according to the data type, computation purpose, dataset contents, and the security concerns of data providers, they make a decision and calculate each parameter.

These numbers are the trade-off between the accuracy and privacy of the dataset. We have fully reviewed the effect of epsilon and sensitivity on the privacy and accuracy of our case studies in chapter 5.

All the computation providers should do now is to run the Parmanix command or runs the shell that contains the command. Figure 4.5 shows the command to run the computation in Parmanix.

We have defined all the values and privacy parameters in the "MapReduceConfiguration" class. User must specify Parmanix parameters in the correct order: **“MapReduceEntryPoint”**, **“Dataset name”**, **“UntrustedMapper”**, **“TrustedReducer”**, **“Epsilon”**, **“MinRange”** and **“MaxRange”**, **“MaxKeyPerGroup”**, **“N”** and **“n”**.

Example 1.

As an example, we use a dataset named “BigShop”. This dataset contains 50000 records of sold items in a shop. Table 4.3 shows one record of this dataset.(Note that Bigshop is a dataset only for practical purposes.)

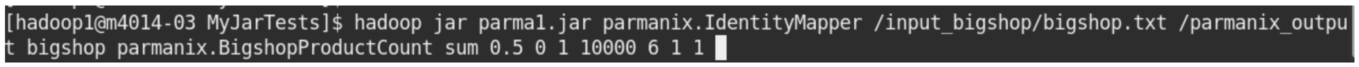
Table 4.3 A record of BigShop dataset

Product Name	Date	Day	Price
XboxOne	2018-04-23	Monday	1.27

We assume the data provider aims to know" how many XboxOne has been sold in the year 2018". The computation provider will write his Mapper code and define keys and identity groups. In the next step, he can compile the source code. The computation provider specifies Parmanix parameters. Table 4.4 shows all the parameters and runs the computation (Figure 4.5)

Table 4.4 Bigshop query parameters

Dataset Name	Bigshop
UntrustedMappeClassName	BigshopProductCount
TrustedReducer	Count
Epsilon	0.5
MinRange	0
MaxRange	1
MaxKeyPerGroup	6
N	1
n	1



```
[hadoop1@m4014-03 MyJarTests]$ hadoop jar parma1.jar parmanix.IdentityMapper /input_bigshop/bigshop.txt /parmanix_outpu
t bigshop parmanix.BigshopProductCount sum 0.5 0 1 10000 6 1 1
```

Figure 4.5 Parmanix Command Execution

We set the minimum and maximum of the range between 0 to 1 as for "Count" a product can be sold or not sold and their existence would make a maximum difference of 1. We have defined 6 for maxKeyPerGroup, as BigShop dataset contains 6 different products including “Ipad”, “Laptop’, “XBoxOne”, “Printer”, “Monitor” and “PS4” so we expect 6 keys. As we expect XBoxOne’s total sale as the output, we expect one key as output, so we chose 1 for N. We assumed that each customer buys only one product so we can set n as 1. The computation provider also needs to define his input path, which is where the dataset has been placed, and output path which is the location where results will be saved.

After defining the parameters, the computation provider will be able to retrieve the computation answer. Without Differential Privacy, the output will be 7152 which is the real answer.

Through Parmanix due to the noise that has been added we have 7098 as the output. Note that the parameters that have been set can be different depending on the priority of privacy and accuracy over each other. This matter will be studied in the next chapter.

CHAPTER 5 EXPERIMENTAL EVALUATION

In this chapter, we demonstrate the practical scenarios and functioning of Parmanix. We would also scrutinize how each parameter of the differential privacy algorithm effects on privacy and accuracy of Parmanix's output. Our purpose is to show how differential privacy protects an individual's data in possible scenarios and also depicts how setting its parameters can affect the outcome. In the end, we implement Support Vector Machine (SVM), a machine-learning algorithm to evaluate the effect of Parmanix on the accuracy of the outputs.

5.1 Experiment on a Real-World Scenario

In order to test the functionality of Parmanix and to show how differential privacy impacts the analysis and protects the individual's data in practice, we have designed a real-world scenario. We have a dataset belongs to an international drug company, "PharmaMedicine", and it is made of the sales records of January 2009. This dataset consists of 1000 records that include 12 columns: Transaction date, Product Name, Price, Payment Type, Customer Name, City, State, Country, Account Created date, Last Login, Latitude and Longitude.

In this scenario, we have a record belongs to a customer, Mr. Smith, that has been diagnosed with cancer and do his drug purchase from this company. He orders "Paclitaxel" on a monthly basis. Paclitaxel is a drug-related to cancer that interferes with the ability of cancer cells to divide. This is not a drug that the company sells an unlimited number to customers but in a regular timely manner like a monthly basis. So, Mr. Smith orders it each month.

In the dataset, we have 3 products: "Triprolidine", "Acetaminophen" and "Paclitaxel". Except the latter, the other two are related to cold and can be sold in an unlimited quantity.

Mr. Smith is living in the region Lombardy of Italy. Recently, Mr. Smith has moved from Lombardy to a new neighborhood. Hence, what we have here is the drug company sale datasets of various months that include Mr. Smith as a customer in Lombardy who buys Paclitaxel each month, but on the other hand, on the January sale dataset, he does not exist as one of the Lombardy customers, because he is not living in Lombardy before. What is the threat in this scenario? How the information about Mr. Smith can be at the risk of revealing?

There might be an adversary, who wants to make sure Mr. Smith truly has cancer. This adversary can be someone who wants this info for an insurance company that Mr. Smith in his new region of living wants to register. This person can be from an advertising company that works with the PharmaMedicine company to do their advertisement. So, to make business decisions for their ads, they can be in contact with the data analytic company that works with PharmaMedicine and give them info like how many percents of customers are from different countries and regions, or which products have fewer sales, etc. As you can see, violation of privacy can be this complicated and unintentional by companies but like this scenario, they can lead to revealing information about people.

In the datasets, there are various columns of data that give many options to the adversary to get to Mr. Smith's information. The adversary can try to query the number of customers who bought products belongs to the specific latitude/longitude (Lombardy) that Mr. Smith was already living. He can do this for months of September, October, November, December of 2008 and then January 2009.

Paclitaxel is not a drug like the other two cold drugs that so many people and in a high rate order. As a result, the adversary can compare the results of Paclitaxel sold item numbers in Mr. Smith old neighborhood before and after he moved from the area and by observing that there is a difference in the output numbers, get to this conclusion that Mr. Smith is the customer who does purchase Paclitaxel and have cancer.

A note here is important; In the real world, the adversaries might possess some previous knowledge from external means that helps them to infer an individual's secret. Such as Netflix example that we mentioned before, which attackers had access to the IMDB dataset and by using both Netflix and IMDB datasets, succeeded to reveal individuals' record data.

In Mr. Smith's scenario, this might be someone who knows that Mr. Smith regularly commutes to the hospital of the city and is a PharmaMedicine customer. This is a sample of knowledge that an adversary can have and according to that suspects and retrieve more data from datasets and according to those data, get to this conclusion that Mr. Smith is a cancer patient and eventually reveal personal information of the customer.

Now we try to implement the same scenario through Parmanix. In the scenario we mentioned above the adversary can compare datasets belongs to various months. However here, to keep things simple and as we want to scrutinize the results, we use only January dataset, once with Mr. Smith in it and another time without him in the dataset.

We have access to latitude and longitude of Lombardy through the dataset records. So, we use it and narrow down our query to the number of sold items in the place to Lombardy which is the place where Mr. Smith was previously living. First, we run the experiment on regular Hadoop (with no differential Privacy). In the dataset that contains Mr. Smith records, the true answer is 6 people from the latitude and longitude that Mr. Smith lives, has bought Paclitaxel.

Our output indicates 6 too.

Now we remove Mr. Smith's records and try the same experiment with Regular Hadoop. The query is the same as before and the dataset is the same but minus Mr. Smith record in it. This time we receive 5 as the output answer. This clearly states that one record does not exist.

We repeat the same experiment with Parmanix. Same as before, we run the same query which is the number of sales per geographical location. First, we run the experiment with Mr. Smith's record in it. We repeat the experiment 5 times and chose:

$\epsilon = 0.5, N = 1, n = 1, 0 < range < 10$

Table 5.1 contains the output Results. In Figure 5.1, we made a Laplacian diagram out of the outputs. The Laplace diagram helps us to depict the distribution of the outputs. In order to make this diagram we repeated the query 50 times.

Table 5.1 Total Number of Paclitaxels Sales Per Geographical Location (Mr. Smith Record In The Dataset)

Execution Number	Total Paclitaxels
1	6
2	5
3	7
4	7
5	5

Now we run the query without Mr. Smith's record in the dataset. The parameters would be the same as before and we repeat it 5 times. Table 5.2 and figure 5.1 (the orange diagram)show the results and the Laplace diagram of this execution. Same as before, to make this diagram we repeated the query 50 times.

It can be seen that with Differential Privacy, someone cannot find out what the real answer is, and this is what we expected. However, what differential privacy does, is taking the certainty from the adversary.

Table 5.2 Total Number of Paclitaxels Sales Per Geographical Location (Without Mr. Smith Record In The Dataset)

Execution Number	Total Paclitaxels
1	6
2	4
3	6
4	5
5	4

We make a comparison between the Laplace diagrams of the query outputs, when Mr. Smith was in the dataset and when he was removed from the dataset. Figure 5.1 shows the aggregation of two diagrams. If you notice the curve of the Laplace diagrams, they roughly are the same. This says that no matter if we add or remove an individual's data record from the dataset (Mr. Smith in this scenario), the output shows the same pattern. In other words, the existence or non-existence of Mr.Smith does not affect the final result and this is exactly what we expect from a differentially private platform.

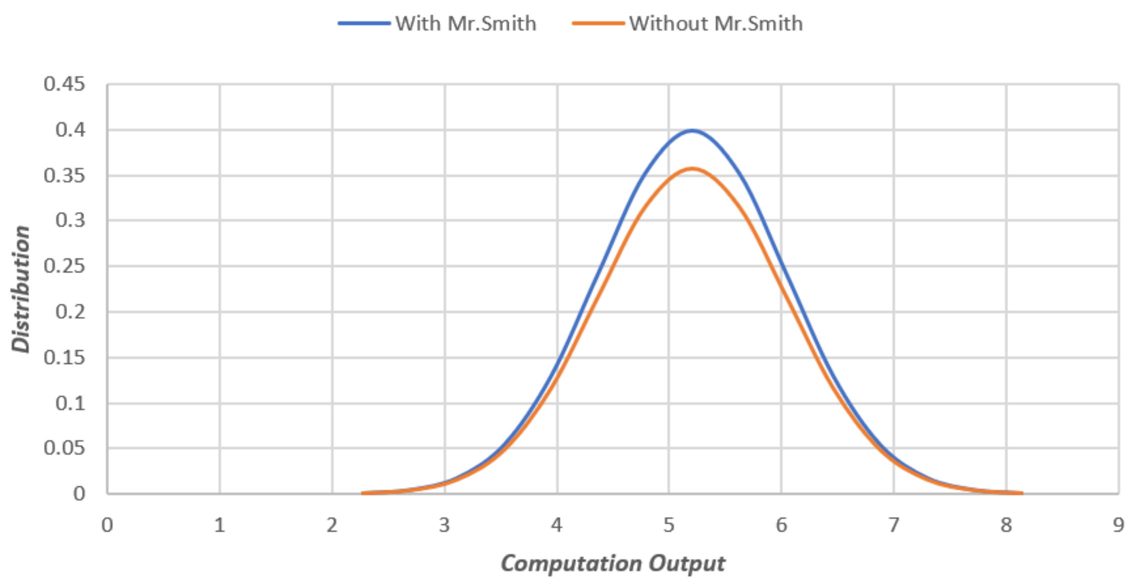


Figure 5.1 Aggregated Results of Computation Output With And Without Mr. Smith In The Dataset

The output of this scenario shows the approximate number of customers of each product in that specific area. This is still enough and helpful for data providers to get to the conclusion that how much sales they have for their product on that geographical location. Thus, they are still capable of making their business decisions according to these results.

The main goal is using data to improve decision making, have a better understanding of the market, training machine learning algorithms, etc. and at the same time not letting that people's data be at risk of exposure and affecting their lives.

Through this scenario, we depicted a simplified real-world scenario and showed how differential privacy protects an individual's data.

5.2 Accuracy and Privacy Trade-off

There are parameters that must be considered in differential privacy algorithms. The importance of these parameters is due to the effect they have on privacy as well as the accuracy of the output results. There is still no rigorous method to control the trade-off between the strength of the "privacy guarantee" and the "accuracy" of the output results. Eventually, it is important to know how they affect the output results and based on that, make the crucial decision of choosing them.

In this section, we present real-world scenarios and scrutinize the effect of these parameters on the accuracy and privacy of the output results.

Epsilon

Epsilon (ϵ) is one of the parameters that affect the privacy of data. It is a metric of privacy loss at a differentially change in data. In fact, epsilon is the central parameter that controls the strength of the privacy and eventually the accuracy. In real-world situations, with the same value of epsilon, the privacy guarantees that are enforced by differential privacy would be different, as the domain of attribute, sensitivity, and the quarry can be different from one situation to another situation.

In order to demonstrate the effect of epsilon, we run an experiment with Parmanix. Same as the previous scenario, we use the sales dataset of PharmaMedicine company. As mentioned before, this dataset contains records of sales in the month of January of an international drug company. The products are drug names. We assume the data provider wants to know the income of selling Paclitaxel.

The computation provider defines keys and values and writes the Mapper. The key here is "product". So, the computation provider will define key in the Mapper he is writing. But for the Reducer, he is allowed to use only the secure reducers of Parmanix. Following this, he will use the "Sum" which is Parmanix secure reducer.

We already know that there are 136 Paclitaxels in the dataset and the price of each Paclitaxel is 3600$. So, the ground truth of this query is 489600$ for Paclitaxel. We assume computation provider chose $\epsilon = 0.9$

The output of the first run is 489600$. As it was expected, the noise that has been added do not reveal the true number. In the mentioned example, if I had chosen a different ϵ, a different amount of noise would be added to the final output. To illustrate the effect of ϵ, we run the experiment for $\epsilon = 0.1$, ϵ=0.5 and ϵ=0.9 . For each epsilon, we repeat the experiment 100 times. As before the query is the income of selling Paclitaxel drugs.

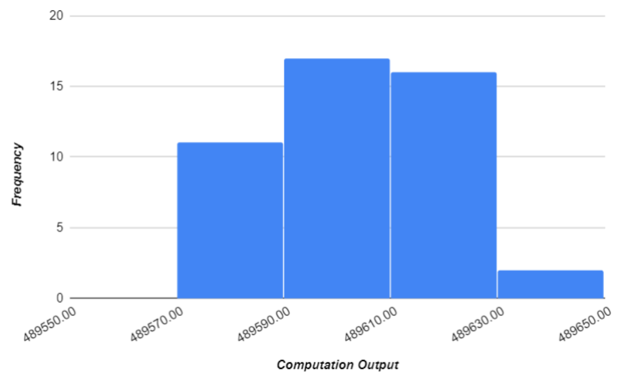
As described in the previous chapters, in Parmanix, the implemented differential privacy adds Laplace-distributed random noise to the data. We have plotted the results of each ϵ to depict the Laplace diagram for each query. But what a Laplace diagram actually shows in this scenario?

Through a Laplace diagram which is known as a double exponential distribution, the distribution of the output can be shown which is usually denser close to the real answer and decreases and scatters as it gets away from the real answer. The parameter that effects on the density or scatteredness of the diagram is epsilon.

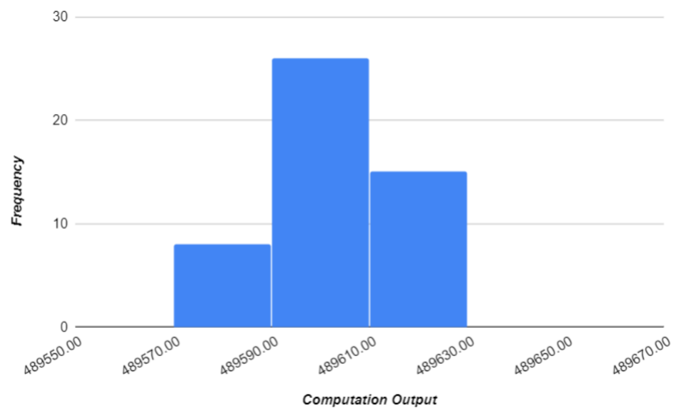
Figure 5.2 shows the histogram and figure 5.3 shows the Laplace distributions of our computation results for $\epsilon = 0.1$, ϵ=0.5 and ϵ=0.9 .

As can be seen through the diagrams, by decreasing the epsilon, more noise would be added to the actual answer(489600) and as a result, the output would be more and more distant from the actual answer. While when we have higher epsilon, less noise would be added and as it can be seen in Laplace distribution diagram, the number of outputs in the ranges close to the true answer is considerably more.

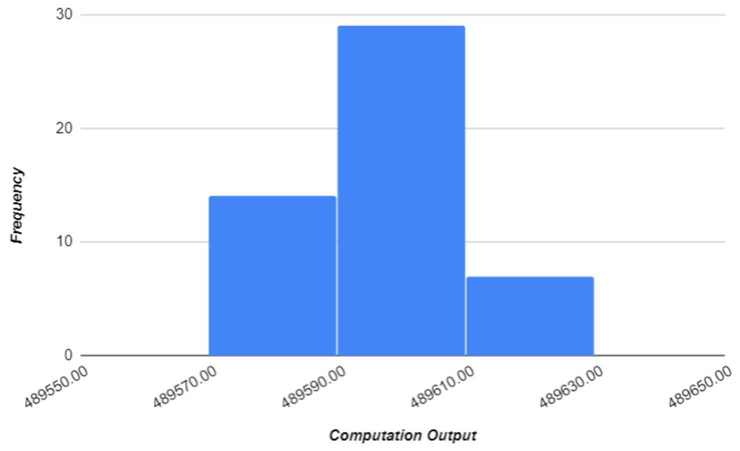
To compare the Laplace diagrams, belong to different epsilons, we have aggregated all 3 Laplace diagrams in figure 5.3. It can be seen, as the epsilon increases, the curves of the diagram decreases. We can observe a less distribution for outputs of $\epsilon = 0.9$ in comparison with $\epsilon = 0.1$; In lower epsilons, the outputs are less aggregated toward the true answer. As said, less epsilon means more noise so as epsilon decreases, more noise will be generated and eventually the output will have more difference with the true answer. This guarantees privacy however higher amount of noise will decrease the accuracy. Thus, we can say a decrease in ϵ will add more noise to the actual data which leads to a decrease in the accuracy.



(a) $\epsilon = 0.1$



(b) $\epsilon = 0.5$



(c) $\epsilon = 0.9$

Figure 5.2 Output Frequency For Different Epsilon Values

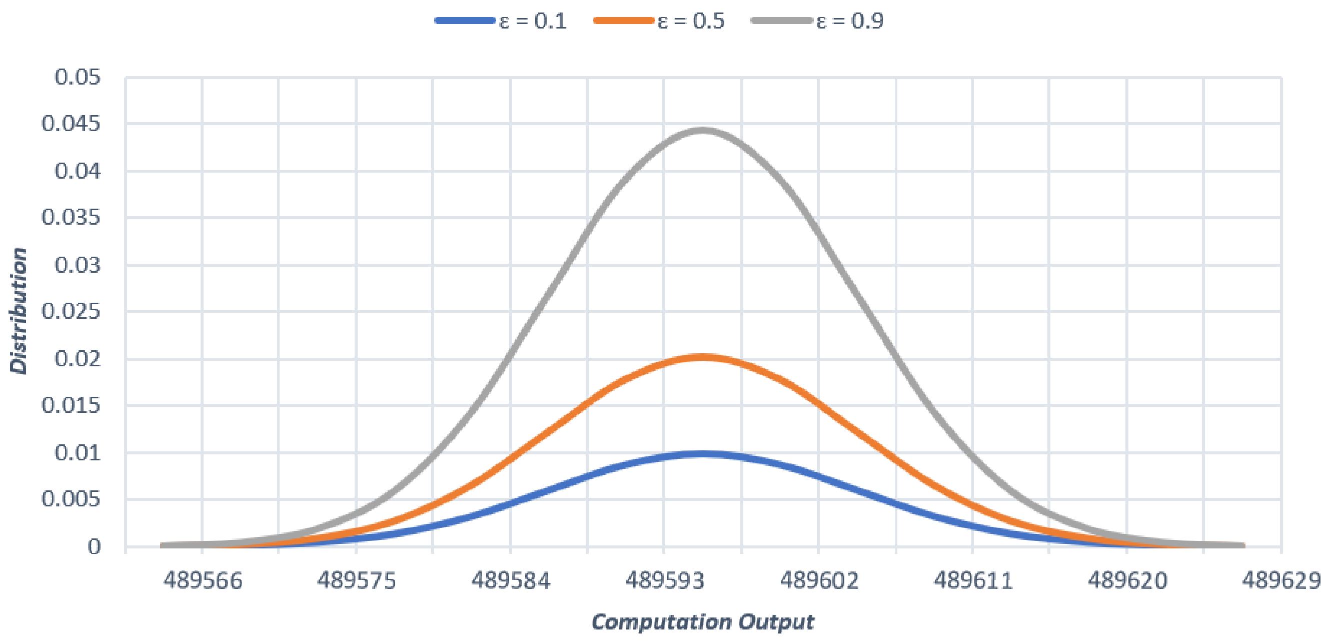


Figure 5.3 Aggregated Laplace Distribution for ϵ= 0.1, ϵ=0.5 and ϵ=0.9

If the algorithm is 0-differential privacy, it protects privacy well, however, it has very low accuracy, then it would be useless! Because we get nothing other than the noise.

ϵ= 0 is equivalent to absolute privacy. According to the definition of differential privacy, it is equivalent to $Pr[M(D)] = Pr[M(D')]$, which leads to this inference that algorithm M is independent of the data and thus perfectly protect privacy.

ϵ controls the crucial trade-off between the strength of the privacy guarantee and the utility of the output results and in particular, there is no rigorous method for choosing ϵ. The Epsilon parameter captures how an individual contribution is hidden.

The biggest debate in differential privacy is this question that how to choose the privacy parameter epsilon? How much noise would be enough?

Setting the proper value of ϵ to protect individuals in the database with some fixed probability is one of the important challenges as it requires balancing the interests of two parties that have conflicting objectives: the data analyst, who wishes to learn something about the data, and the prospective participant, whom his personal data included in the analysis and has the concern of revealing private data.

Sensitivity

Epsilon is not the only controversial parameter of differential privacy; Sensitivity is another factor that affects the privacy and utility of differential privacy output.

We would be able to increase the privacy and protect the data better if we increase the variance of the added noise, but the truth is that we can not simply add any Laplace noise to a function. Each function has its own sensitivity and the amount of the noise that we want to add is dependent on the function's sensitivity.

As mentioned earlier, the sensitivity of a function can be defined as the largest possible difference that one row can have on the result of that function and this is true for any dataset. A real-valued function sensitivity expresses the maximal change that is possible to happen in its value due to the addition or removal of a single record. But how we determine what is THIS largest possible difference?

Finding out the sensitivity is not something the data analyzer seeks inside the dataset alone, but it should be determined according to what the dataset describes. Depending on the whole scenario, what the analyzer wants to take out from the data, the function that he uses and also with the help of the dataset content, the sensitivity can be determined.

To see the effect of sensitivity on the output and the way it affects accuracy and privacy, we have designed another scenario to run with Parmanix. Once again, we use the dataset belongs to PharmaMedicine the international drug company. The dataset contains information regarding the sales in January 2009. In this dataset, we have 3 products.

Table 5.3 List of PharmaMedicine Products And Their Overall Sale

Product	Number of Sold Items	Price	Overall Sale
1- Triprolidine	847	1200	1,016,400$
2- Paclitaxel	136	3600	489,600$
3- Acetaminophen	16	7500	120,000$
			1,626,000$

Table 5.3 shows the actual number of each product in the dataset and the overall sale that the company had during January. This time, what we want is the overall sale of products 1 to 3 in Canada. Thus, we write the related Mapper and Reducer in Parmanix; we use the secure reducer "Sum". For epsilon, we chose $\epsilon = 0.9$ and run the experiment 50 times. Note that the table 5.4 is the actual numbers for Canada.

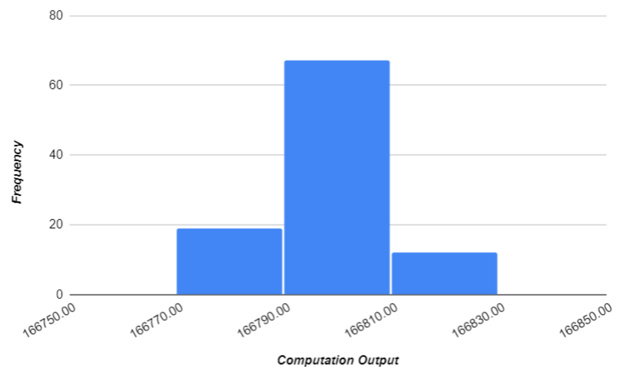
We run the experiment with 2 different sensitivity, low and high but we do not make any changes in epsilon which is 0.9. Parmanix calculates the sensitivity based on range. Eventually to change the sensitivity we have changed the range manually. We set the range between <0, 7500> for a low sensitivity experiment. We used <0 , 20,000> as the range for high

Table 5.4 List of PharmaMedicine Products And Their Overall Sale In Canada

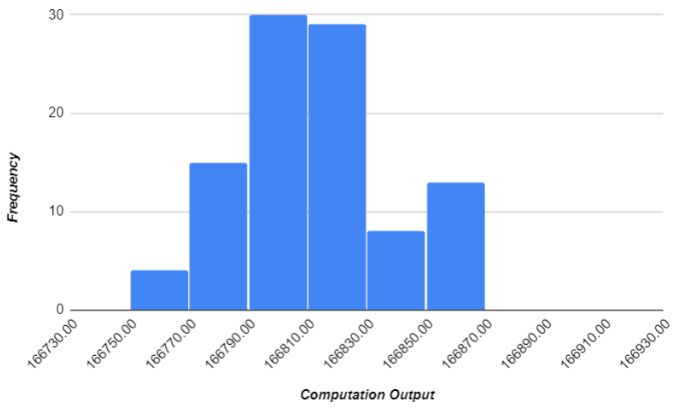
Product	Number of Sold Items	Price	Overall Sale
1- Triprolidine	64	1200	76,800$
2- Paclitaxel	0	3600	0$
3- Acetaminophen	12	7500	90,000$
			166,800$

sensitivity. With the increase in range, the sensitivity increase as well.

We repeated the experiment 100 times; Figure 5.4 and figure 5.5 show respectively the histograms and the aggregated Laplace diagram the experiment. As can be seen, the algorithm needs to add a lot of noise to cover the high sensitivity of the computation. In fact for the same epsilon, the amount of added noise is higher. The curves of the Laplace diagrams show a change in the output pattern. So, the function with higher sensitivity has less curve.



(a) Low Sensitivity



(b) High Sensitivity

Figure 5.4 Output Frequency For Low And High Sensitivity

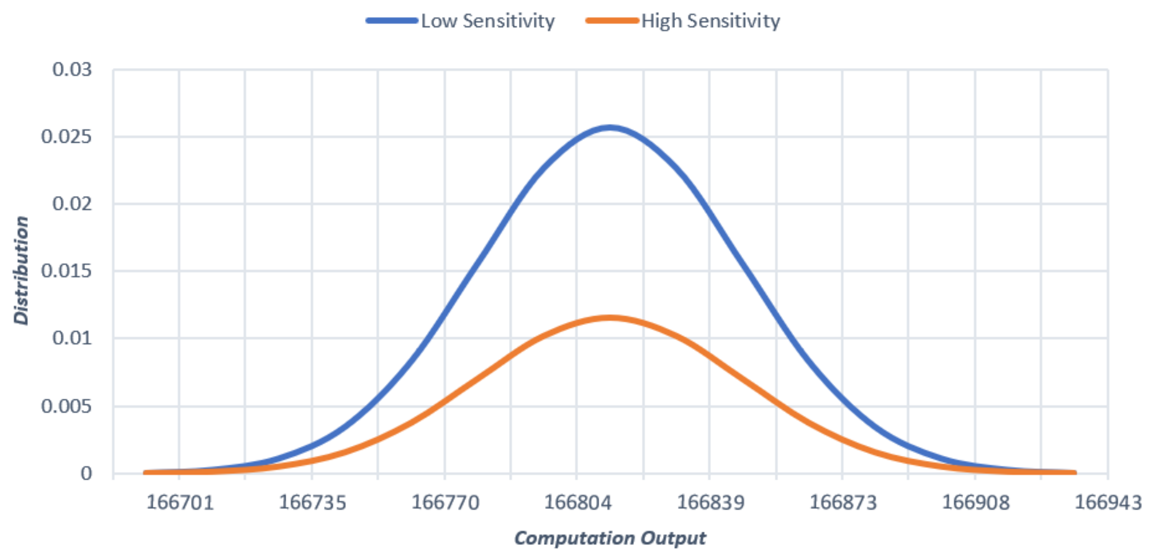


Figure 5.5 Distribution of Outputs With Low And High Sensitivity

In a real-world scenario, we could add only one item with a high different price. This would change the privacy of the sum query even though it was only one item! To keep privacy, adding or removing one item must not change the pattern, if this happens, the privacy has been violated. To cover a highly sensitive dataset, we need to add more noise to avoid information leakage; Therefore, there is a straight relation between the sensitivity of a function and the amount of noise that should be added; the higher the sensitivity, the more random noise should be added.

The amount of noise defines the accuracy of the results. Here both of the functions are sum but the sensitivity for them is different. The first one has lower sensitivity which means less noise. So, the output has higher accuracy. But for the second one, the situation is different; Sensitivity is high, and a lot of noise should be added so the accuracy is lower. This function might keep the promise of privacy and protect an individual's data, but the output results are no longer useful.

The truth is that the functions with higher sensitivity require more noise to obtain a fixed ϵ of privacy. The experiment results clearly show this truth. The epsilon and sensitivity have an opposite impact on the results. While the lower epsilon leads to more noise, lower sensitivity leads to less noise. This is why we talk about the trade-off between the parameters that affect privacy and accuracy in the algorithm.

One of the weaknesses of differential privacy is the effect that sensitivity has on it. As a matter of fact, in the real world, sensitivities are usually so high and calculating it for a function is not usually easy. so, it is unavoidable to face with sensitivity issues. A considerable amount of research effort has spent on solutions to estimate the sensitivity [46] improving the methods to estimate it and even circumventing the calculation of sensitivity [47]. As a conclusion, what the data providers and computation providers should consider when it comes to data analysis, is knowing the effect of these parameters and make a trade-off between the accuracy and privacy according to what they want to extract from the data.

5.3 Utility and Performance

Parmanix, is a cloud-based module, runs on top of Hadoop. One of the important factors for us was that while Parmanix is running, it has minimal effect on Hadoop execution which means creates minimum overhead and eventually it presents an acceptable performance.

To test the utility of Parmanix, we have designed three experiments. We will run each experiment on the original Hadoop and Hadoop with the Parmanix module. We aim to compare the average runtime for each platform and calculate how much is the overhead of Parmanix and how it affects the performance of regular Hadoop. In fact, we want to know if using Parmanix in data analytics is something that worth the efficiency costs it imposes?

The Hadoop version is 3.2.1 and runs on the Intel Core i7 CPU 2.93 GHz machine with 8 GB RAM. In general, when running computation in Hadoop, the server needs to be running and needs restarting on every change in configuration. The data sets are stored on the internal solid-state drive of the system and the good thing about it is that solid-state drives do not have idling issues that conventional hard drives have.

In our experiments, we have considered the size of the dataset as well as the function that we want to run on the dataset.

We have used 3 datasets. They have the same columns, but the difference is in their sizes. The small one has only 100 records, the medium one has 50,000 records and the third has 500,000 records. The datasets belong to a hypermarket and contain information regarding the products and sales. Table 5.5 shows all 14 columns of the datasets.

Table 5.5 Dataset Attributes

Region	Country	Item Type	Sales Channel	Order Priority	Order Date	Order ID
Ship Date	Units Sold	Unit Price	Unit Cost	Total Revenue	Total Cost	Total Profit

We want to know the total revenue of the “Office Supplies” in 2014. In this scenario, we assume $\epsilon = 0.9$. The function which secure Reducer of Parmanix will use is "Sum". To calculate the execution time, the tests that were executed were pushed to the JobTracker; we made some changes in the “TaskLog” file to provide the time; no batch scripts or automatically generated CSV files were involved.

To start, we run the test for the small dataset; We should consider that in this dataset, the total number of “Office Supplies” that have been sold is 12 and the actual answer for revenue is 30585380.07$

First, we run the experiment with regular Hadoop. As it was expected, the output is 30585380.07$. We executed it for 5 times and calculated the average runtime. The next step is to run it on Parmanix. We repeated the experiment 5 times and the same as before, calculated the average runtime of each execution.

Now, we repeat the experiment with the medium dataset which includes 50,000 records. Same as the previous experiment, what we want is the total revenue of the “Office Supplies”. Finally, we run the same experiment on the large dataset of 500,000 records.

Figure 5.6 shows the comparison of the runtimes in Hadoop and Parmanix. As can be seen for a small dataset with the size of 100 records, the runtime difference is in the scale of milliseconds which is so fast and as can be seen, there is not much difference (approximately 10% overhead) in the runtime of Hadoop and Parmanix.

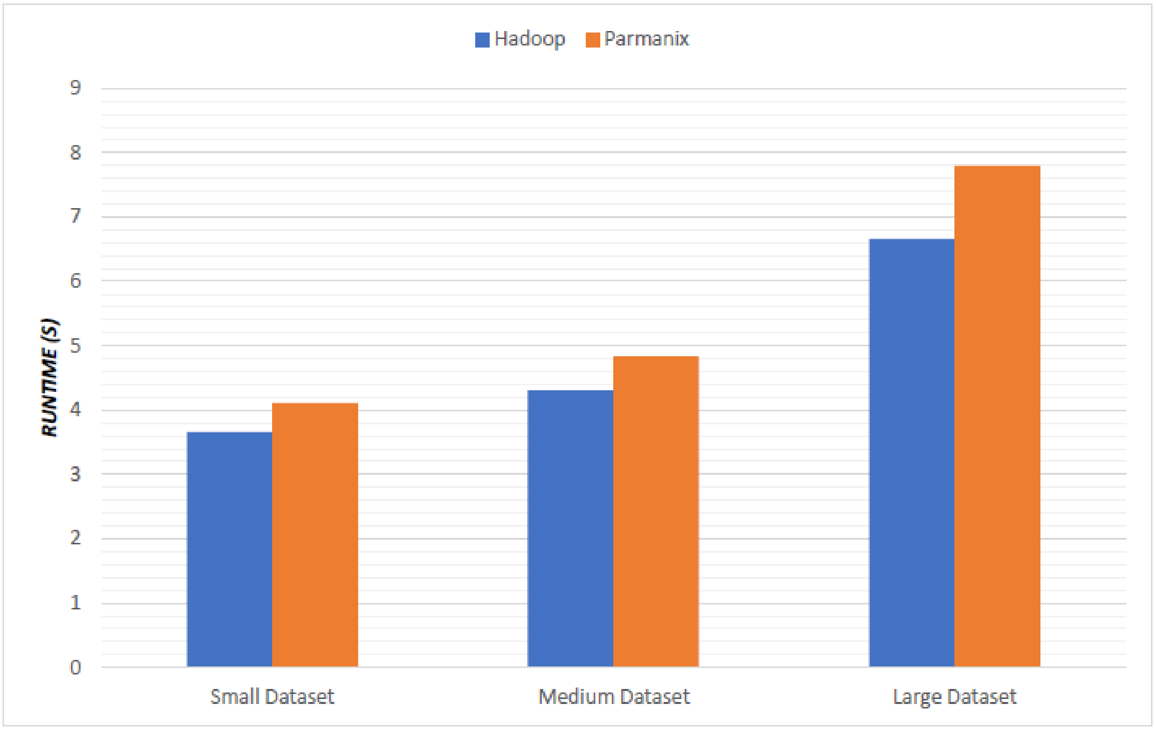


Figure 5.6 RunTime Comparison Between Hadoop And Parmanix with 3 Different Dataset Size

As can be seen in this figure, with an increase in dataset size, more time is needed to do the computation, however, Parmanix runtime is close to Hadoop. To show this better figure 5.7 shows the overhead that Parmanix has imposed on the system.

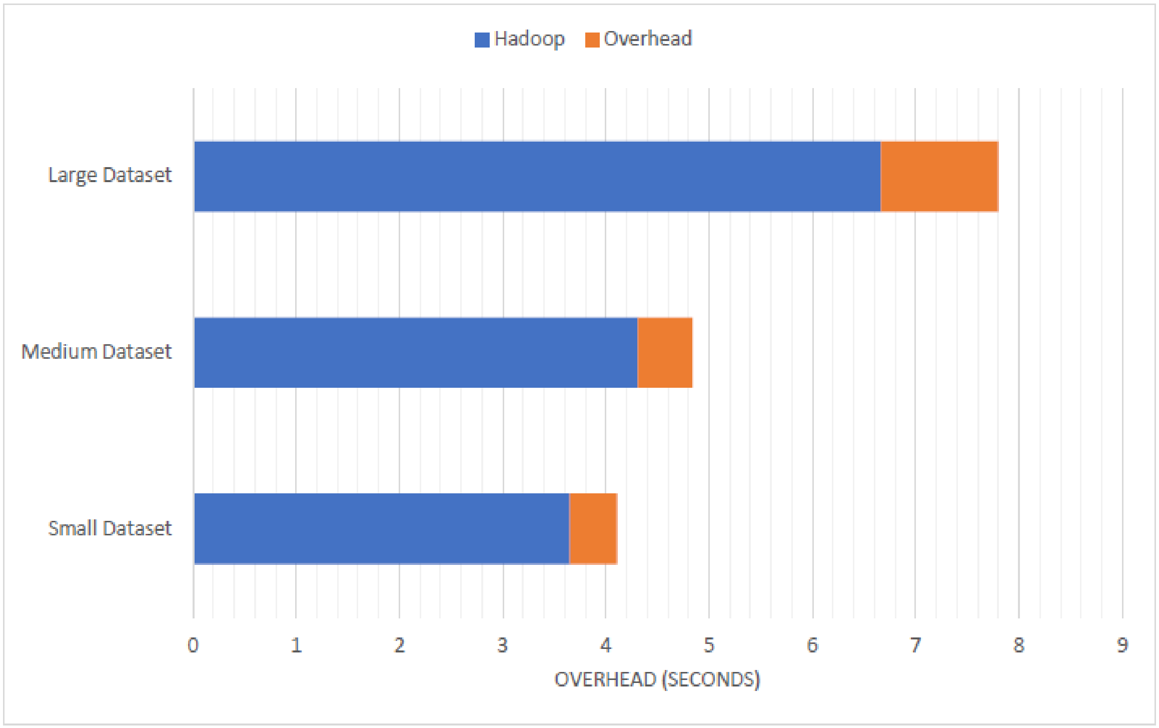


Figure 5.7 Imposed Overhead to Hadoop By Parmanix

The overhead percentage is almost the same for the small and medium dataset (approximately 12%). This increases to 17% for Large datasets. With contemplating the size of datasets and run time, it is safe to claim the overhead is still acceptable. Figure 5.8 shows the overhead percentage comparison between Hadoop and Parmanix.

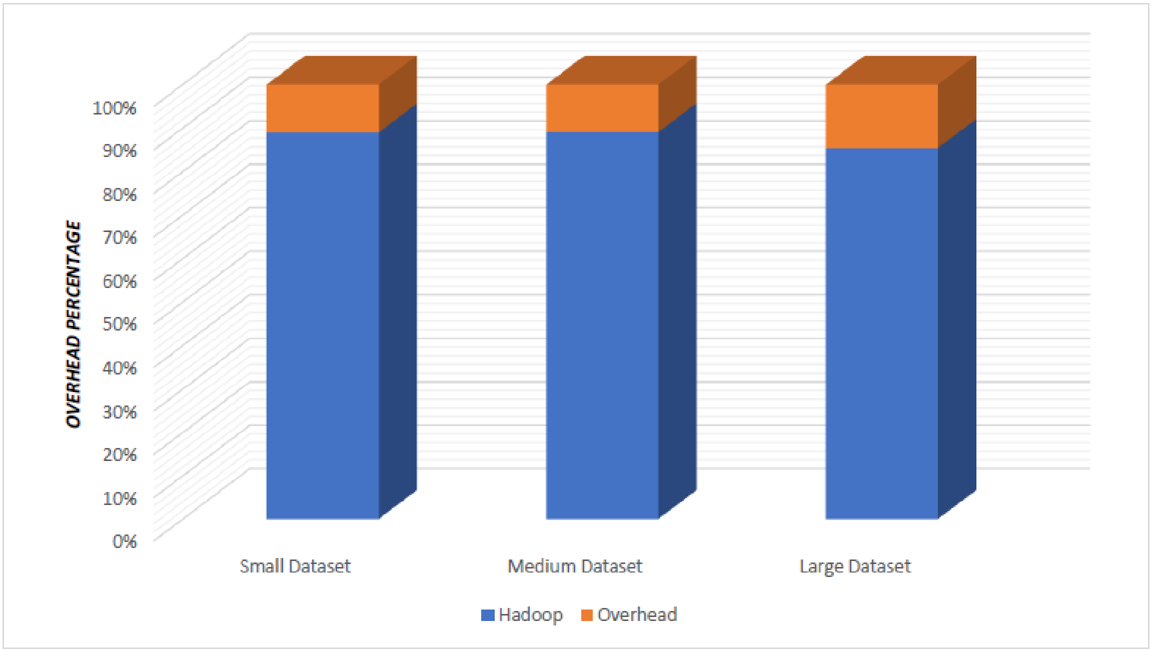


Figure 5.8 Overhead Percentage Comparison In Hadoop And Parmanix

Note that when running the tests, although the configuration was set to have more active nodes for the smaller datasets only 1 node was active at the time. The reason for this behavior appeared to be the block size of the HDFS in which the data files are stored. By default, this block size was set to 64 MB, so the data sets of 10 and 50 MB could fit into a single block, eventually allowing only one node to work on it. To change this, we set the block size of the HDFS to 1 MB. By this change, blocks of 1 MB were created and as a result, the smaller datasets could be executed by multiple nodes.

In the IT world, any security and privacy solution affects the utility and has its own overheads that slow down the process. At the same time, it is inevitable to provide security and privacy for the data; Especially in data analytics that a lot of individuals' data are at the risk of exposure and the side effects this has on people's lives are critical. So, it is important the security and privacy solutions offer an acceptable utility and performance. In the real world, the size of datasets is high, and this increases the need for faster platforms. Hadoop Map-Reduce has been designed to feed the need for faster parallel environments that be able to answer data analytic demands as the traditional platforms failed to fulfill these demands. Eventually, the privacy solution that we add to it, should not have a contradiction with the goal of Hadoop which is a faster analysis of big data.

This is something that we tried to provide in Parmanix as the test results show roughly positive feedback regarding achieving this goal. Hadoop Developers during the years have added functions to improve performance. A possible reason for acceptable performance in Parmanix is avoiding touching the main source code of Hadoop.

Implementation of Parmanix on Sparks can be another solution to reduce the overhead even less than this. Sparks is an open-source distributed cluster-computing framework and has been designed for faster analytics than Hadoop MapReduce.

In some ways, the architecture of Sparks is different that makes it be a faster platform in comparison with Hadoop. Spark has in-memory processing which means there would be no time spent on moving data/processes in and out of the disk; Also, Unlike Hadoop, Spark does not come with its own file system; instead, it can be integrated with many different types of file systems such as Hadoop's HDFS, Amazon's S3 system, MongoDB, etc.

Implementation of Parmanix on Spark can be an interesting topic and we leave it for future works.

5.4 Classification Algorithm

In order to evaluate Parmanix, we used it to compute a machine learning algorithm, Support Vector Machine (SVM), to observe its accuracy loss compare to the original Hadoop.

SVM is a supervised machine learning technique. Its main use is for classification and regression and is defined by a separating hyperplane as by finding suitable hyperplanes that are able to separate the data by the highest margin it attempts to categorize data. Finally based on the training sets it segregates and analyzes the new values [48].

In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. In two dimensional space, this hyperplane is a line dividing a plane into two parts wherein each class lay on either side. The goal of the SVM is to design a hyperplane that classifies all training vectors in two classes. In fact the best choice of hyperplane is the one that leaves the maximum margine from both classes.

To do this experiment, we used the 20newsgroup dataset [1]. This dataset is a collection of approximately 20,000 newsgroup documents, that consists of different articles represented by words that appear in them and have been partitioned across 20 different newsgroups. The 20 newsgroups dataset is commonly used for experiments in text applications of machine learning techniques, such as text classification.

[1]http://qwone.com/ jason/20Newsgroups/

We used this dataset and performed WordCount function in Parmanix. The mapper takes the labeled data and outputs category and word as <key, value> pair. The Reducer will sum up all the words of each category and outputs the WordCount pair as <key, value>. We turned the output of WordCount function into "Tf-Idf" format. Tf-Idf or frequency-inverse document frequency is a statistical measure and is often used in information retrieval and text mining. It evaluates the relevance of each word to a document in a collection of documents by multiplying the times a word appears in a document and also multiplying the inverse document frequency of the word across the collection of the documents.

We used this data as input to the SVM classifier. Our data is 2-dimensional as it consists of documents and words. We split the data into a training set and test set.

In our experiment we used a 5-fold cross validation and by this we partitioned the data into 5 subsets and ran the process 5 times, each time one subset was on hold while the rest of them were used for training. Eventually each time we used 80% of the data for training and kept 20% for testing. The outcome of this execution on the training set, is a model that will be evaluated and its performance would be validated on the test dataset.

Note that since we wanted to use multi-class for SVM we used the "one-vs-rest" approach to create multiple hyperplanes (one for each category).

As mentioned earlier, the goal of this experiment is to evaluate the fluctuations of accuracy against the privacy parameter epsilon in a machine learning approach. Higher epsilons means less noise would be added to the output result and we aim to calculate how much this affects the accuracy of classification. The accuracy of the classifier would be calculated by the number of misclassified articles.

We have executed the tests 5 times and calculated the average percentage. As figure 5.9 shows, for $\epsilon = 0.2$ the accuracy of classification is 48% while this increases to 72% for $\epsilon = 1.4$. As a result, the increase in the epsilon will lead to higher accuracy of the classification

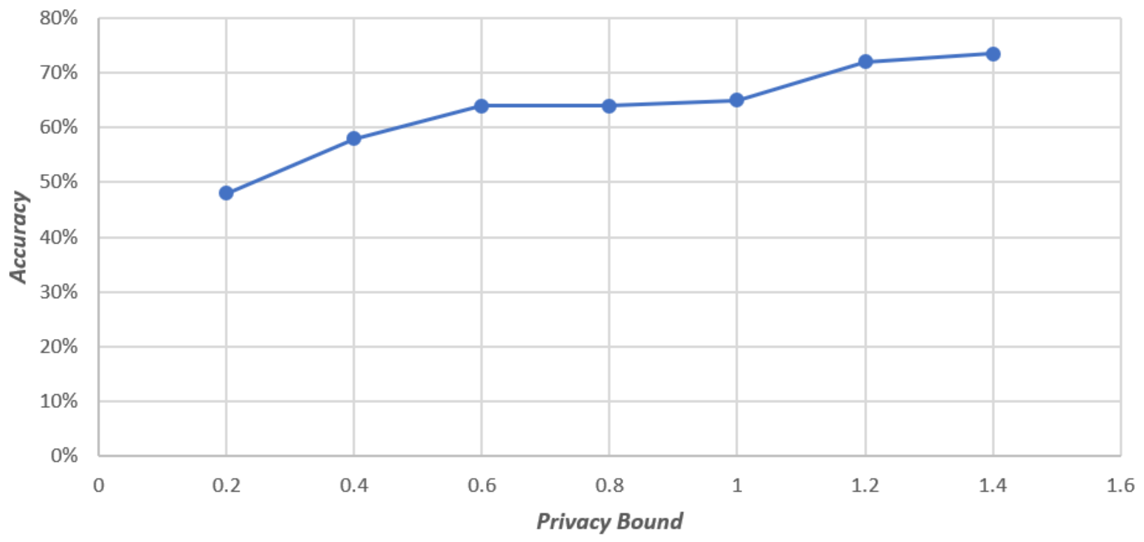


Figure 5.9 Accuracy In SVM

CHAPTER 6 CONCLUSION

6.1 Summary

In this study, we have proposed a distributed computation module based on MapReduce that guarantees the privacy of an individual's data. Parmanix integrates differential privacy algorithm into Hadoop MapReduce and makes sure that for any possible input, the computation output does not depend on the presence of that input in the data set. This privacy preserve module, by calculating the proper amount of noise, that keeps the balance between the privacy and accuracy of the final output, guarantees the privacy-preserving.

The Mapper can be insecure as the platform has been designed to not leak data through Mapper. For the Reducer phase, secured Reducers have been implemented and the computation provider must use them instead of writing his own Reducer code. Parmanix confines the computations and prevents information leakage beyond the data provider's policy.

Parmanix has been inspired by Airavat, however the source code of Parmanix has been written from scratch and unlike Airavat, we have not made any changes in the Hadoop source code. Unfortunately, we did not have access to the main source code of Airavat and eventually, we were not able to make a comparison between Parmanix and Airavat performance. The results of the case study show how epsilon and sensitivity, which are the main privacy parameters in Parmanix can affect the output results.

6.2 Limitations of Our Approach

Data providers usually want an absolute privacy guarantee, independent from the nature of computations carried out on the data. The fact is such a thing is not possible and unfortunately, an absolute privacy guarantee cannot be achieved for meaningful definitions of privacy. As [49] showed, the entire dataset can be decoded with a linear number of queries. Parmanix has its limitations too. Limitations that in the future works can be improved but never can cover every aspect of security and privacy of data.

One of the limitations is that the module can not confine every computation of untrusted code. One of these computations is key generating. Mapper generates keys however we know that mappers are untrusted. If the mapper is malicious, it can use the string nature of the keys and provide a storage channel that later leaks data. Parmanix has no guarantee for this. That is why the computation provider must provide a key list and leave the rest of the work (value-generating for the declared keys) to the platform.

6.3 Future Works

Compromise between security and performance is always a discussion in data security and privacy. Implementation of the current framework on Apache Spark which does the processes in memory can considerably increase the performance of this system. In future works, we will consider migrating Parmanix to spark.

Parmanix guarantees the privacy of individual's data however adding access control techniques will guarantee data leakage through system resources. Adding access control solutions to Parmanix can considerably improve MapReduce security and Privacy.

REFERENCES

[1] K. M. P. Shrivastva, M. Rizvi, and S. Singh, “Big data privacy based on differential privacy a hope for big data,” in *2014 International Conference on Computational Intelligence and Communication Networks.* IEEE, 2014, pp. 776–781.

[2] D. D. Hirsch, “The glass house effect: Big data, the new oil, and the power of analogy,” *Me. L. Rev.*, vol. 66, p. 373, 2013.

[3] C. Dwork *et al.*, “Calibrating noise to sensitivity in private data analysis,” *Journal of Privacy and Confidentiality*, vol. 7, no. 3, pp. 17–51, 2016.

[4] C. Dwork, “Differential privacy: A survey of results,” in *International conference on theory and applications of models of computation.* Springer, 2008, pp. 1–19.

[5] M. Yang *et al.*, “Personalized privacy preserving collaborative filtering,” in *International Conference on Green, Pervasive, and Cloud Computing.* Springer, 2017, pp. 371–385.

[6] G. S. Bhathal and A. Singh, “Big data computing with distributed computing frameworks,” in *Innovations in Electronics and Communication Engineering.* Springer, 2019, pp. 467–477.

[7] G. Bhathal and A. Singh, “Big data: Hadoop framework vulnerabilities, security issues and attacks,” *Array*, vol. 1-2, p. 100002, 07 2019.

[8] Which companies are using hadoop for big data analytics? [Online]. Available: https://kognitio.com/big-data/companies-using-hadoop-big-data-analytics/

[9] K. Abouelmehdi *et al.*, “Big data security and privacy in healthcare: A review,” *Procedia Computer Science*, vol. 113, pp. 73–80, 2017.

[10] S. Desai *et al.*, “Improving encryption performance using mapreduce,” in *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems.* IEEE, 2015, pp. 1350–1355.

[11] R. R. Parmar *et al.*, “Large-scale encryption in the hadoop environment: Challenges and solutions,” *IEEE Access*, vol. 5, pp. 7156–7163, 2017.

[12] P. Goswami and S. Madan, "Privacy preserving data publishing and data anonymization approaches: A review," in *2017 International Conference on Computing, Communication and Automation (ICCCA)*. IEEE, 2017, pp. 139–142.

[13] N. Victor, D. Lopez, and J. H. Abawajy, "Privacy models for big data: a survey," *International Journal of Big Data Intelligence*, vol. 3, no. 1, pp. 61–75, 2016.

[14] What is mapreduce. [Online]. Available: https://www.talend.com/resources/what-is-mapreduce/

[15] D. Cynthia, "Differential privacy," *Automata, languages and programming*, pp. 1–12, 2006.

[16] C. Dwork, A. Roth *et al.*, "The algorithmic foundations of differential privacy," *Foundations and Trends® in Theoretical Computer Science*, vol. 9, no. 3–4, pp. 211–407, 2014.

[17] A. Wood *et al.*, "Differential privacy: A primer for a non-technical audience," *Vand. J. Ent. & Tech. L.*, vol. 21, p. 209, 2018.

[18] O. O'Malley *et al.*, "Hadoop security design," *Yahoo, Inc., Tech. Rep*, 2009.

[19] D. Das *et al.*, "Adding security to apache hadoop." hortonworks report," 2011.

[20] S. Y. Ko, K. Jeon, and R. Morales, "The hybrex model for confidentiality and privacy in cloud computing." *HotCloud*, vol. 11, pp. 8–8, 2011.

[21] K. Zhang *et al.*, "Sedic: privacy-aware data intensive computing on hybrid clouds," in *Proceedings of the 18th ACM conference on Computer and communications security*, 2011, pp. 515–526.

[22] T. D. Vo-Huu, E.-O. Blass, and G. Noubir, "Epic: efficient privacy-preserving counting for mapreduce," *Computing*, vol. 101, no. 9, pp. 1265–1286, 2019.

[23] F. Armknecht *et al.*, "A guide to fully homomorphic encryption." *IACR Cryptology ePrint Archive*, vol. 2015, p. 1192, 2015.

[24] A. Khaled *et al.*, "A token-based access control system for rdf data in the clouds," in *2010 IEEE Second International Conference on Cloud Computing Technology and Science*. IEEE, 2010, pp. 104–111.

[25] Z. Kaoudi and I. Manolescu, "Rdf in the clouds: a survey," *The VLDB Journal*, vol. 24, no. 1, pp. 67–91, 2015.

[26] H. Ulusoy *et al.*, "Vigiles: Fine-grained access control for mapreduce systems," in *2014 IEEE International Congress on Big Data*. IEEE, 2014, pp. 40–47.

[27] P. Ulusoy *et al.*, "Guardmr: fine-grained security policy enforcement for mapreduce systems," in *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, 2015, pp. 285–296.

[28] A. Machanavajjhala *et al.*, "Privacy: Theory meets practice on the map," in *2008 IEEE 24th international conference on data engineering*. IEEE, 2008, pp. 277–286.

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

[30] S. R. Ganta, S. P. Kasiviswanathan, and A. Smith, "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.

[31] M. Kantarcioglu and C. Clifton, "Privacy-preserving distributed mining of association rules on horizontally partitioned data," *IEEE transactions on knowledge and data engineering*, vol. 16, no. 9, pp. 1026–1037, 2004.

[32] F. D. McSherry, "Privacy integrated queries: an extensible platform for privacy-preserving data analysis," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, 2009, pp. 19–30.

[33] I. Roy *et al.*, "Airavat: Security and privacy for mapreduce." in *NSDI*, vol. 10, 2010, pp. 297–312.

[34] R. Millman. (2017) Thousands of hadoop clusters still not being secured against attacks. [Online]. Available: https://www.scmagazineuk.com/thousands-hadoop-clusters-not-secured-against-attacks/article/1475302

[35] B. Zhou and J. Pei, "The k-anonymity and l-diversity approaches for privacy preservation in social networks against neighborhood attacks," *Knowledge and information systems*, vol. 28, no. 1, pp. 47–77, 2011.

[36] H. T. Greely, "The uneasy ethical and legal underpinnings of large-scale genomic biobanks," *Annu. Rev. Genomics Hum. Genet.*, vol. 8, pp. 343–364, 2007.

[37] L. Sweeney, "Recommendations to identify and combat privacy problems in the commonwealth," 2005.

[38] V. Ciriani *et al.*, "κ-anonymity," in *Secure data management in decentralized systems.* Springer, 2007, pp. 323–353.

[39] K. El Emam and F. K. Dankar, "Protecting privacy using k-anonymity," *Journal of the American Medical Informatics Association*, vol. 15, no. 5, pp. 627–637, 2008.

[40] R. J. Bayardo and R. Agrawal, "Data privacy through optimal k-anonymization," in *21st International conference on data engineering (ICDE'05).* IEEE, 2005, pp. 217–228.

[41] K. Purdam and M. Elliot, "A case study of the impact of statistical disclosure control on data quality in the individual uk samples of anonymised records," *Environment and Planning A*, vol. 39, no. 5, pp. 1101–1118, 2007.

[42] A. Narayanan and V. Shmatikov, "Robust de-anonymization of large sparse datasets," in *2008 IEEE Symposium on Security and Privacy (sp 2008).* IEEE, 2008, pp. 111–125.

[43] ——, "How to break anonymity of the netflix prize dataset," *arXiv preprint cs/0610105*, 2006.

[44] C. Culnane, B. I. Rubinstein, and V. Teague, "Health data in an open world," *arXiv preprint arXiv:1712.05627*, 2017.

[45] C. Whong. (2014) Foiling nyc's taxi trip data. [Online]. Available: https://chriswhong.com/open-data/foil_nyc_taxi/

[46] K. Nissim, S. Raskhodnikova, and A. Smith, "Smooth sensitivity and sampling in private data analysis," in *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, 2007, pp. 75–84.

[47] A. Team, "Learning with privacy at scale," *Apple Mach. Learn. J*, vol. 1, no. 9, 2017.

[48] M. Jain *et al.*, "Speech emotion recognition using support vector machine," *arXiv preprint arXiv:2002.07590*, 2020.

[49] I. Dinur and K. Nissim, "Revealing information while preserving privacy," in *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 2003, pp. 202–210.

APPENDIX A

The related codes and datasets can be found in the GitHub address:

https://github.com/Crac2019