Anonymizing Large Transaction Data Using MapReduce

A thesis submitted in partial fulfilment

of the requirement for the degree of Doctor of Philosophy

Neelam Memon

January 2016

School of Computer Science & Informatics
Cardiff University

Declaration

This work has not previously been accepted in substance for any degree and is not concurrently submitted in candidature for any degree.

Signed .... (candidate)

Date ....

Statement 1

This thesis is being submitted in partial fulfillment of the requirements for the degree of PhD.

Signed .... (candidate)

Date ....

Statement 2

This thesis is the result of my own independent work/investigation, except where otherwise stated. Other sources are acknowledged by explicit references.

Signed .... (candidate)

Date ....

Statement 3

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed .... (candidate)

Date ....

Abstract

Publishing transaction data is important to applications such as marketing research and biomedical studies. Privacy is a concern when publishing such data since they often contain person-specific sensitive information. To address this problem, different data anonymization methods have been proposed. These methods have focused on protecting the associated individuals from different types of privacy leaks as well as preserving utility of the original data. But all these methods are sequential and are designed to process data on a single machine, hence not scalable to large datasets.

Recently, MapReduce has emerged as a highly scalable platform for data-intensive applications. In this work, we consider how MapReduce may be used to provide scalability in large transaction data anonymization. More specifically, we consider how set-based generalization methods such as RBAT (Rule-Based Anonymization of Transaction data) may be parallelized using MapReduce. Set-based generalization methods have some desirable features for transaction anonymization, but their highly iterative nature makes parallelization challenging. RBAT is a good representative of such methods. We propose a method for transaction data partitioning and representation. We also present two MapReduce-based parallelizations of RBAT. Our methods ensure scalability when the number of transaction records and domain of items are large. Our preliminary results show that a direct parallelization of RBAT by partitioning data alone can result in significant overhead, which can offset the gains from parallel processing. We propose MR-RBAT that generalizes our direct parallel method and allows to control parallelization overhead. Our experimental results show that MR-RBAT can scale linearly to large datasets and to the available resources while retaining good data utility.

Acknowledgements

I would like to thank all those who have given me academic and moral support for my research work over the last years. I would like to thank the department of Computer Science and Informatics, in particular to my supervisors, Dr. Jianhua Shao and Dr. Grigorios Loukides for their guidance and valuable advice. I would like to thank all the school staff members, specifically my annual review panel members, for their help and constructive feedback to my research. I would like to also thank my family and friends, for their support.

Contents

Abstract **ii**

Acknowledgements **iv**

Contents **v**

List of Figures **viii**

List of Tables **x**

List of Algorithms **xii**

1 Introduction **1**

1.1 Privacy and Its Protection 2

1.2 Privacy Protection in Transaction Data 3

1.3 Research Problem 7

1.4 Research Methodology 8

1.5 Research Challenges 9

1.6 Contribution 10

1.7 Thesis Structure 11

2 Related Work 13

2.1 Transaction data anonymization 13

2.1.1 Generalization-based Transaction Data Anonymization 15

2.2 Non-Parallel Methods for Scalable Data Anonymization 22

2.3 Possible Choices for Parallelization 26

2.3.1 PRAM 27

2.3.2 BSP 27

2.3.3 MapReduce 28

2.4 MapReduce and Privacy Protection 32

2.5 Other applications of MapReduce 34

2.6 Summary 41

3 RBAT and MapReduce 43

3.1 Preliminaries and Problem Definition 43

3.1.1 Problem Statement 47

3.2 Overview of RBAT 48

3.3 Parallel Design of RBAT using MapReduce 51

3.4 Summary 53

4 Direct Parallelization of RBAT 55

4.1 Data Partitioning and Representation 56

4.2 Parallel Support Computation 61

4.3 Direct Parallelization of RBAT 62

4.4 Experimental Evaluation 71

4.5 Summary 82

5 MR-RBAT: A Scalable Transaction Data Anonymization Method 84

5.1 MR-RBAT 85

5.1.1 α-Split 86

5.1.2 γ-Check 91

5.2 Performance Estimation 95

5.3 Experimental Evaluation 99

5.3.1 Evaluation of MR-RBAT 100

5.3.2 Evaluation of Performance Estimation 107

5.4 Summary 109

6 Conclusions and Future Work 111

6.1 Research Summary 111

6.2 Future Work 114

References 116

List of Figures

2.1 An example hierarchy 15
2.2 A example R-tree for k-anonymous data 24
2.3 A example 2-anonymous data using Mondrian 25
2.4 A schematic representation of a MapReduce round 29

3.1 An example Split-Tree 50
3.2 A general framework for MapReduce-based design of RBAT 52

4.1 A schematic representation of Parallel RBAT 63
4.2 Datasize vs. Runtime 73
4.3 Scalability with respect to cluster size 75
4.4 Scalability with respect to cluster size 76
4.5 Performance with respect to $|\Theta|, k = 60, c = 0.8$ 77
4.6 Performance with respect to $|\Theta|, k = 80, c = 0.7$ 79
4.7 Rules Input vs. Checked (k = 60, c = 0.8) 80
4.8 Performance with respect to $|I|$ 81
4.9 Scaleup 82

5.1 Effect of α 101

5.2 α vs. Skewness 102

5.3 Scalability with respect to Datasize 103

5.4 Scalability with respect to Cluster Size 104

5.5 Effect of $|I|$ 105

5.6 Effect of γ 106

5.7 Scaleup 107

5.8 Evaluation of cost estimation of α-Split with respect to α 108

5.9 Evaluation of cost estimation of γ-Check with respect to γ 108

List of Tables

1.1 An example transaction dataset 3

1.2 De-identified transaction dataset 4

1.3 An example of set-based generalization 6

2.1 An example dataset 16

2.2 A 2-anonymized dataset using local generalization 16

2.3 A 2-anonymized dataset using global generalization 16

2.4 Comparision of transaction data anonymization methods 23

2.5 An example Dataset T 24

3.1 An example dataset D 44

3.2 The most generalized dataset $\tilde{D}$ 45

3.3 An anonymized dataset $\tilde{D}$ 51

4.1 Notations used in our work 56

4.2 An example dataset T 57

4.3 The default parameter settings for experiments 73

4.4 Hadoop parameter settings 74

5.1 Empirical values for evaluation of cost estimation 107

List of Algorithms

3.1 RBAT (D, Θ, $\tilde{i}$, k,c) 48

4.1 COMPUTEUL (P) 64

4.2 FINDMAXPAIR ($\tilde{i}$) 65

4.3 MR-SPLIT ($\tilde{i}$) 67

4.4 UPDATECUT ($\tilde{P}$, $\tilde{i}$, $\tilde{i}_l$, $\tilde{i}_r$) 69

4.5 GENERALIZE (D, $\tilde{P}$) 69

4.6 MR-CHECK (D, Θ, k, c) 71

5.1 MR-RBAT (D, Θ, $\tilde{i}$, k, c) 87

5.2 α-SPLIT ($\tilde{i}$, α) 89

5.3 γ-CHECK (D, $\tilde{P}$ Θ, k, c, γ) 94

Chapter 1

Introduction

The digital data containing person-specific information are increasingly being collected, analysed and disseminated, often in response to government laws or to benefit organizations. For example, private California-based hospitals [13] are required to publicly release some demographic data for every patient discharged from their facility[1]. Electronic Patient Record [88], an online system in the United Kingdom, allows patients' demographics, their medical and billing details to be collected and shared. Tesco, a large retail organization, collect and share their customers' information with a range of retailers and manufacturers [103]. Different countries including the United States of America, the United Kingdom and Canada have also created a range of online data repositories following open government initiatives [90, 120, 91].

These collected datasets are valuable sources for analytical studies. For example, health-related personal information shared among healthcare institutions can help advance personalized medicine [42]. Business Intelligence is another application of person-specific data. As reported by Chen *et al.* [15], survey results showed that 97 percent of companies with revenues exceeding $\$100$ million are engaged in business analytics using the data containing personal information. However, such data often contain sensitive information about individuals, and releasing them in their original

[1]http://www.oshpd.ca.gov/hid/Products/Hospitals/QuatrlyFinanData/CmpleteData/default.asp

form may compromise individuals' privacy. According to a survey [119], 89% of online users have expressed serious concerns about their privacy. Estimated by Forrestor Research, e-tailers lose $15 billion a year due to privacy concerns of online consumers [19]. Westin [61] conducted over 30 surveys from 1978 to 2004 to show people's attitude towards privacy over time and found that the majority fall into the Pragmatic category *i.e.* they are willing to share their information only with trusted parties. It is important therefore that we consider how individuals' privacy contained within data may be protected.

1.1 Privacy and Its Protection

Despite the fact that many attempts have been made over a long time [28, 107], we do not have a single, universally agreed-upon definition for privacy. For example, Westin [128] defined privacy as an ability to know when, how, and to what extent information about us is communicated to others. Parent [95] defined it as the amount of control someone has over information about himself. Allen [4] defined it as an ability to control the way others have access to us. Not surprisingly, how privacy may be protected will depend on the specific aspects of privacy that one seeks to protect. For example, cryptography [118, 134] and access control [10, 47] seek to provide secrecy and protect access to information, whereas data anonymization [34] controls the amount of information that can be publicly released.

In this thesis, we consider a scenario where data are required to be released to third parties or to be made available publicly. Typically, how the published data may be analysed is not known at the time the data are published. For example, in October 2006, Netflix released a dataset containing 100 million anonymous movie ratings and challenged the research community to develop predictive algorithms that would do better than their own recommendation system [6]. At the time of publishing the data, Netflix could not be certain how the data might be used or analysed. To protect privacy in this type of scenario, we need to have the data sanitized (anonymized) to a pre-defined privacy level, so that no sensitive information about the individuals contained in the data will be disclosed when the data are released.

1.2 Privacy Protection in Transaction Data

Data can be of different forms, for example, relational data such as demographics, graph data produced by social networks, search logs maintained by search engines, and location data collected by location-based applications. Among them, transactional data are of interest and important. They are a major source of information for understanding and analysing the habits and activities of an individual.

A transaction dataset is a collection of records called transactions. Each transaction is a set of items drawn from a domain and is assumed to be associated with an individual. Table 1.1 shows an example of transaction dataset containing query terms posed by search engine users, where each search term is an item and each row (collection of search terms) is a transaction.

User	Search logs
Jerry	Flu symptoms, Dutch floral designs, **Depression and medical leave**
Alice	Fiber mark , Car crash photos, **How to kill your wife**
Viccie	Parking near Georgia town, Numbness in hand, **Breast cancer**
Jackie	Parking near Georgia town, Dogs food, **Breast cancer**

Table 1.1: An example transaction dataset

Transaction data are useful in different applications. For example, search logs such as the dataset given in Table 1.1 can help personalized web search. The diagnosis records and market basket data can assist in personalized medicine and purchasing trend projection. However, transaction data also contain person-specific sensitive information, as can be seen from Table 1.1, which can be disclosed when the data are released.

To protect such data, transactions are typically de-identified prior to their release. That is, direct person identifying items such as social security numbers, names or phone numbers are removed. Table 1.2 shows the de-identified transactions given in Table 1.1. Unfortunately this may not provide sufficient protection for individuals' privacy because combinations of non person-identifying items may still be used to identify an individual in a de-identified dataset [5, 84]. For example, AOL released the de-identified search logs of its $650,000$ users in 2006 in order to support research [5]. Searcher No. 4417749 was tracked back to Thelma Arnold, a 62-year-old resident of Lilburn from the released dataset by her neighbour, even if the explicit identifiers such as user names and IP addresses were removed from the released search records. Netflix, the world's largest online movie rental service, released a de-identified dataset containing movie ratings records of nearly a half-million of its users in 2006, with the motive to improve the accuracy of its movie recommendations algorithm. Narayan *et al.* [85] used the rating information from Internet Movies Database (IMDb), another movies-related website to show that only a small fraction of external information (rating dates and some movies) is needed to identify Netflix subscribers even if the published data are fully de-identified. This can then lead to the disclosure of the sensitive information associated with them, by observing and inferring from movie titles suggesting political beliefs (*e.g.* Fahrenheit 9/11), religious views (*e.g.* Jesus of Nazareth), and sexual orientations (*e.g.* Queer as Folk).

	Search Logs
	Flu symptoms, Dutch floral designs, **Depression and medical leave**
	Fiber mark , Car crash photos, **How to kill your wife**
	Parking near Georgia town, Numbness in hand, **Breast cancer**
	Parking near Georgia town, Dogs food **Breast cancer**

Table 1.2: De-identified transaction dataset

Publishing transaction data therefore requires protection against two types of potential privacy leak: identity disclosure and sensitive information disclosure. *identity disclos-*

ure occurs when an individual can be linked to their transactions in the dataset. This usually happens when certain combination of items occurs uniquely or infrequently within the released data and an attacker has the knowledge of this combination. For example, if we know that Viccie is a Lilburn resident and has a numbness problem, then releasing Table 1.2 will lead to the unique identification of her record in the dataset. *Sensitive item disclosure* is where sensitive information about an individual is learnt with or without identifying their transactions. For example, knowing that Jackie has searched for Parking near Georgia Town, one can infer that she has searched for Breast Cancer too.

One solution to preventing such disclosures is to anonymize data [21, 36]. That is, we perform some form of data transformation so that the frequency of item occurrences is altered, hence the two types of disclosure are prevented in the transformed data. Different anonymization techniques have been proposed in the literature [113, 116, 50, 12, 36, 132, 73]. A popular approach is set-based generalization which attempts to hide an original item by replacing it with a set of items. For example, Table 1.3 is a set-based generalized version of Table 1.2, where *(Fiber mark, Numbness in hand, Dogs food)* is a generalized item which is interpreted as representing any non-empty subset of the items contained in it. As can be seen, with this type of generalization, Viccie can no longer be uniquely identified from Table 1.3, even if we know that she has a numbness problem.

However, protecting transactions through set-based generalization has a price to pay: we lose some utility in the generalized data. For example, suppose that we need to count the number of times the search terms Fiber Mark and Dog Foods appear together in the search log. Using Table 1.3, three transaction records satisfy this query whereas in the original dataset (Table 1.1), it is not supported by any transaction record. It is important therefore to have an anonymization method that will not lead to excessive data distortion. However, finding an optimal generalization solution that achieves the required privacy protection and retains the utility of the data as much as possible is

	Search Logs
	Flu symptoms, Dutch floral designs, **Depression and medical leave**
	(Fiber mark, Numbness in hand, Dogs food) , Car crash photos, **How to kill your wife**
	Parking near Georgia town, (Fiber mark, Numbness in hand, Dogs food), **Breast cancer**
	Parking near Georgia town, (Fiber mark, Numbness in hand, Dogs food) **Breast cancer**

Table 1.3: An example of set-based generalization

challenging.

Existing transaction data anonymization approaches [113, 116, 50, 12, 132, 73] are designed to perform anonymization in sequential settings, and typically assume the use of a single centralized machine. This limits their applicability to small-scale datasets. However, data are becoming larger and more scalable anonymization methods are needed.

One way to enhance the scalability of existing methods is to optimize their memory utilization. For example, using sampling [63, 65, 75] can help reduce memory requirement. However, such strategies may still not be able to deal with increasingly large datasets [80]. Sample sizes can still be very large and the anonymization solutions derived from the samples can significantly compromise privacy and data utility. For instance, Walmart handles more than a million customers every hour, collecting an estimated 2.5 petabytes of data in the process [1]. Gartner predicts that the amount of data in all forms will grow 650% in the next five years [3]. According to IDC[2], the amount of digital information doubles every eighteen months. In 2011, the data held by hospitals and medical centres were estimated to be about one billion terabyte [3]. When dealing with such large-scale data, parallel and distributed computing using

[2]http://www.idc.com

shared-nothing commodity clusters is becoming a promising direction.

Recently, MapReduce [22] has emerged as a scalable and cost-effective data-processing platform, providing a simple, yet powerful parallel computing paradigm. In recent years there has been an increasing support for MapReduce from both industry and academia, making it one of the most rapidly adopted frameworks for parallel processing of large data volumes today [58]. Google uses its implementation of MapReduce to process more than ten petabytes of information per day [22, 58]. Hadoop, an open source implementation of MapReduce, has been adopted by more than 160 different organizations including commercial companies and research institutes such as Yahoo, Twitter, Facebook, Amazon and IBM[3].

1.3 Research Problem

Our work aims to study how MapReduce may be used for large transaction data anonymization. More specifically, we attempt to address the scalability issues associated with set-based generalization methods [75, 69, 73]. We will focus on the parallelization of RBAT (Rule-Based Anonymization of Transaction data) [72], an existing transaction data anonymization method, on MapReduce.

RBAT is able to deal with both types of disclosure and can retain high data utility by allowing fine-grained privacy requirements to be specified. Moreover, it does not require a generalization hierarchy, hence its search space for generalization solutions is wider than those explored by hierarchy-based methods. This allows better utility to be retained in anonymized data.

Since, achieving required privacy level and good data utility are important aspects of a data anonymization method, our study is based on the following hypothesis:

A MapReduce-based parallelization of RBAT will scale linearly to large transaction

[3]http://wiki.apache.org/hadoop/PoweredBy

data while achieving the same privacy level and utility of anonymized data as its sequential counterpart.

Note that while we consider the parallelization of RBAT using the MapReduce paradigm in this thesis, some design aspects of our proposed method are applicable to other transaction data anonymization methods too. Thus, our solutions are not limited to RBAT only.

1.4 Research Methodology

Recall that this study aims to check the feasibility of the application of MapReduce in design of a transaction data anonymization solution that is scalable to large data and achieves the required level of privacy without significant loss of utility. We perform our study in the following sequential steps.

1. The first and foremost step towards the design of scalable transaction data anonymization is to design a partitioning strategy. RBAT has more than one inputs that can be considered for partitioning. There can be more than one way to partition an input and to represent the partitions in the memory of the assigned nodes. Also, different partitioning methods can have different effects on the anonymized results as well as the level of parallelization that can be achieved. Therefore, we design a partitioning method by studying different possible partitioning methods and identifying their effect on the anoymized results and on the level of parallelization that can be achieved.

2. Next, we design the parallelelization of RBAT. To carry this out, we first identify the high-level steps that must be kept intact to ensure that the same anonymized results can be achieved as RBAT does. For each step, we design its parallel implementation by partitioning data and performing the key operations using MapReduce.

3. Finally, we experimentally evaluate our implementation over a physical cluster. We mainly measure the scalability with respect to large problem instances and the physical

resources available. We also analyse the performance and identify any bottlenecks that may have arisen due to the design constraints, imposed by MapReduce.

1.5 Research Challenges

Parallelizing a set-based generalization method such as RBAT on MapReduce is challenging, and the following issues need to be tackled:

1. Data Partitioning is important to any algorithm that attempts to process a large amount of data in parallel and it is important to the parallelization of RBAT too. Partitioning must allow to deal with large problem instances in a scalable way and the workload among processing nodes must be balanced. Transaction datasets often are large and contain varying numbers of items in each record. Therefore, even when the number of records are equally split among partitions, the workload associated with each partition may not be fairly distributed. In MapReduce, parallel tasks are created typically by partitioning one input and all other inputs need to be replicated to each processing node [23]. Dealing with large problem instances consisting of large transaction records or the domain of items is challenging.

2. Most of the computations in set-based generalization methods such as RBAT require a global view of data. A MapReduce computation usually consists of a sequence of two key computational stages: Map and Reduce. The communication among processing node during these stages is not allowed except at the end of the map stage. If the computations assigned to a processing node are dependent on the data assigned to the other nodes, most of the data will need to be communicated to the reduce stage via network interconnection. Communicating such a large amount of data can incur high communication cost and may result in ineffective resource utilization. Therefore careful partitioning and placement of input is required.

3. RBAT is a good representative of iterative transaction data anonymization. The method iterates until the solution gives the required privacy level with best possible utility based on some heuristics. Each iteration consists of a number of iterative steps. To ensure good utility of data, these steps must be performed in sequence. In MapReduce, no data is kept between any iterations. Each iteration requires to set up parallel tasks and to re-load data associated to the tasks. Designing a performance-effective solution is not straightforward.

4. Parallelization may also affect the utility of anonymized data. If the scalability achieved is at the cost of sacrificing a large amount of data utility, then the anonymized results may not be usable, rendering our algorithm ineffective.

1.6 Contribution

In this work, we address the challenges described in the previous section. In particular, we make the following contributions:

1. We propose a partitioning scheme for transaction data. The method does not put any restriction on the number or size of any partitions and allows to balance the workload associated with each partition even when the datasets consist of transaction records of varying sizes. It partitions more than one input and hence allows to deal with problem instances consisting of a large number of transaction records or domain of items. This is useful for achieving scalability. Our partitioning does not require a large amount of data to be communicated among machines and therefore does not incur high network cost. Our partitioning scheme also represents data in a way so that the required computations are performed efficiently.

2. We propose two parallelizations of RBAT: direct parallelization and loop-controlled parallelization. The direct parallelization implements RBAT on MapReduce by

mainly partitioning data in order to achieve scalability. Our experimental study shows that the method is scalable but its performance can be even worse than its sequential counterpart due to large parallelism overhead. The loop-controlled parallelization deals with the parallelism overhead and addresses the performance bottlenecks of our direct parallel method. Our method is based on the idea that increasing the number of MapReduce rounds increases the level of synchronization during anonymization operations but decreases the communication cost at each MapReduce round, whereas decreasing the number of rounds to be employed will incur less parallelism overhead but increases the amount of computations during each MapReduce round. Our loop-controlled method generalizes the direct parallel method and allows to control the number of MapReduce rounds to be employed at each iteration of the anonymization process.

3. We evaluate our design on real-world transaction datasets using a commodity cluster. Our experimental results show that our loop-controlled method can scale linearly to large datasets, and nearly linear to cluster sizes. Our method can perform nearly 9 times faster than RBAT using a cluster of 14 processing nodes while retaining almost the same level of utility as RBAT does.

1.7 Thesis Structure

The rest of the thesis is organized as follows:

Chapter 2 surveys the existing work relevant to our problem of large-scale transaction data anonymization. We briefly discuss the existing transaction data anonymization methods, and review different techniques used to address the scalability issues of data anonymization in centralized settings. The application of MapReduce for privacy-preserving data publishing and other areas are also surveyed.

Chapter 3 gives the background necessary to understanding our work. We define relevant concepts and introduce some important notations. This is followed by a brief overview of RBAT and MapReduce. We also present a framework where general issues concerning the parallelization of a method such as RBAT on MapReduce will be discussed.

Chapter 4 describes a direct parallelization of RBAT using MapReduce. We give detailed description of our data partitioning and representation mechanism, and our MapReduce-based design for RBAT. We experimentally evaluate the scalability of our method over large data volumes of up to 128 million real-world transaction records using a commodity cluster consisting of 14 homogenous slave nodes and a master node.

Chapter 5 analyses in detail the performance bottlenecks that arise from the direct parallelization of RBAT presented in Chapter 4. We present our loop-controlled parallelism to deal with these performance bottlenecks. We also present the details of the solution to deal with these performance bottlenecks. We give an analytical model to estimate the performance of our proposed method. We test the scalability of our method on various data volumes and available resources.

Chapter 6 concludes the thesis and presents some possible future research directions for our work.

Chapter 2

Related Work

This chapter will review the existing literature relevant to our problem of anonymizing large transaction data. We first review the existing works on transaction data anonymization, then techniques used by existing sequential methods to address the problem of scalable anonymization. We also study the application of MapReduce in different applications including privacy preservation methods.

2.1 Transaction data anonymization

There are considerable research efforts for designing privacy-preserving data publishing methods [36, 2]. Privacy-preserving data publishing aims to protect the privacy of individuals whose records are contained in the released data. One way to achieve this is to anonymize data, before releasing the data to untrusted recipients. The anonymization techniques typically transform the data in a way that the published records are protected against identity and sensitive information disclosure attacks [34].

Existing data anonymization methods mostly differ in two important aspects: privacy protection and utility preservation. A method must ensure that the anonymized data is protected against the disclosure of identity and the sensitive information contained in their records. But this must not be at the cost of high data utility loss. A method that gives maximum possible privacy protection but distorts the data significantly, may result in the published data far from serving the purpose of its release. Therefore, some tradeoff between privacy and utility must be made. The privacy and utility level achieved by an anonymization method is determined by the privacy model and the anonymization techniques used.

A privacy model defines the constraints of privacy protection. For example, k-anonymity [110] requires that the probability of an individual being uniquely identified in the released data is no more than $\frac{1}{k}$. Other privacy models include l-diversity [76], t-closeness [68], δ-presence [86], (c,t)-isolation [14], (a,k)-anonymity [129], (c, l)-diversity, confidence bounding [125, 124] and ϵ-differential privacy [26, 25]. These privacy models cannot be trivially adopted to anonymize transaction data, as transaction records do not follow a small fixed schema and each record is a subset of items usually drawn from a large domain. So applying these privacy principles to transaction data may cause high information loss, which can render the anonymized data useless [39, 75]. Therefore, a different set of privacy models, considering specifically the characteristics of transaction data, were proposed. For example, k^m-anonymity [114], (h, k, p)-coherence [132], complete k-anonymity [51], and ρ-uncertainty [12].

An anonymization technique is used to transform data in order to satisfy a privacy model. Several data anonymization techniques exist: generalization [35, 109, 126, 56], suppression [109, 56], perturbation [100, 18] and dissociation [40, 117]. Generalization [35, 109, 126, 56] transform data by replacing a value with a more generalized but consistent value. Suppression [109, 56] anonymizes data by removing a value or a record. Perturbation [100, 18] distorts the data by adding noise, swapping values, or generating synthetic data based on some statistical properties of the original data. Dissociation [40, 117] dissociates the relationship between sensitive and non-sensitive items. These techniques have implications on the privacy and utility achieved. For example, suppressing the values or the records gives high privacy protection but may also cause high data distortion. Perturbation preserves statistical information and the anonymized data can be used to build data mining models accurately but noise addition falsifies the information contained at record-level. Hence, the anonymized results are limited in their applications. For example, when biomedical studies [72, 34] require analyses to be made at record-level, such perturbed data may not be useful. In this thesis, we consider the methods based on generalization since they provide good utility of data and retain the original information at record-level. In the following, we will discuss these methods and the level of privacy and utility they provide in detail.

2.1.1 Generalization-based Transaction Data Anonymization

Different generalization-based methods exist for transaction data anonymization [72, 114, 132, 51, 115, 70, 12, 73]. These methods achieve the required privacy by replacing an item with its generalized representation containing less specific information. Generalization may be performed by allowing different occurrences of an original item to be mapped to different generalized items. Such generalization is called Local Generalization. For example, consider 2-anonymizing the dataset shown in Table 2.1.

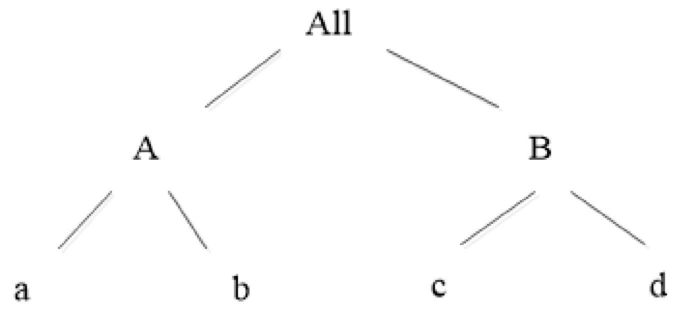


Figure 2.1: An example hierarchy

Using the hierarchy shown in Figure 2.1, the anonymized form of Table 2.1 using local generalization is shown in Table 2.2. Note that the occurrences of items a and b are intact in the last three transaction records whereas the same items are replaced by their generalized representation A in the first two transaction records.

Global generalization forces all the occurrences of an item to have only one generalized

TID	Items
1	a
2	a, b
3	a, b, d
4	a, b, c, d
5	a, b, d

Table 2.1: An example dataset

TID	Items
1	A
2	A
3	a, b, B
4	a, b, B
5	a, b, B

Table 2.2: A 2-anonymized dataset using local generalization

representation. For example, 2-anonymized form of Table 2.1 using global generalization is given in Table 2.3.

TID	Items
1	A
2	A
3	A, B
4	A, B
5	A, B

Table 2.3: A 2-anonymized dataset using global generalization

Below, we survey some local and global generalization methods for transaction data anonymization.

Local Generalization-based Methods

He *et al.* [51] and Terrovitis *et al.* [115] proposed local-generalization based methods to achieve complete k-anonymity and k^m-anonymity.

He *et al.* [51] proposed a top-down greedy partitioning method, which uses local generalization to achieve k-anonymity for transaction data. The method starts with all the transaction records creating a single partition. It then splits each partition into sub-partitions, until no split can be performed without violating the required privacy protection. Logically, each partition represents a generalized representation, corresponding to a level in given generalization hierarchy. Every time when a partition is to be split, one of the generalized item associated with the partition is specialized to its child nodes in the hierarchy. The choice of the generalized item is made greedily, depending on which specialization causes maximum information gain. This requires iterating over all the generalized items associated with the partition and computing the information gain as a result of specialization. The method terminates when no partition can further be split without violating the privacy protection requirement.

Terrovitis *et al.* [115] proposed a local recoding method, which sorts the records and partition them into n parts. Each part is anonymized independently using their proposed Apriori anonymization method to achieve k^m-anonymity. The whole process generalizes the dataset progressively in m iterations. That is, the i_{th} iteration ensures that the anonymized data achieves k^i-anonymity. This is done by computing the number of occurrences of all itemsets of size i and finding the unprotected itemsets in given datasets. For each itemset found unprotected, the algorithm attempts to find a generalized representation that makes the itemset protected and causes minimum information loss. To make the count of occurrences of the itemsets efficiently and reducing memory requirements, the algorithm scans the whole dataset at the beginning of each iteration and constructs a count tree, similar to a FP-tree [48].

Local generalization potentially causes less data distortion than global generalization.

However, the inconsistent generalization of items in a dataset may result in derivation of inaccurate results [34, 75]. Also, all the existing local recoding methods achieve privacy protection against identity disclosure attack and are not trivial to extend them for protection against sensitive itemset disclosure. The methods are also based on the models with some uniform privacy assumptions and therefore may excessively distort the data. For example, complete k-anonymity is based on the assumption the itemsets of all sizes in given dataset need to be protected. Therefore, the model requires that each transaction record must be identical to $(k-1)$ other transaction records. Similarly, k^m-anonymity requires that all itemsets of size m must be contained by at least k transaction records.

The MapReduce realization of such local generalization methods is relatively trivial, since the algorithms recode data chunks independently. However, the issues such as dealing with the large domain of items may still need to be addressed. For example, an i_{th} iteration of apriori anonymization [115] constructs a count tree, containing all possible itemsets of size i and their possible generalizations. Similarly, the partitioning method proposed in [51] requires the taxonomy tree to be loaded into the memory, while anonymizing each partition.

Global Generalization-based Methods

Anonymization methods employing global generalization has also been proposed. Some of these methods find the anonymized solution by replacing one or more items by any of their ancestor nodes in the hierarchy. Other methods do not require any generalization hierarchy and the generalized representation is created dynamically by the methods.

Hierarchy-based Methods

Terrovitis *et al.* [114] designed a global generalization-based method to achieve k^m-anonymity. Their method works in bottom-up fashion. Starting from original items, the algorithm performs m iterations, protecting increasingly large combination of items in each iteration. During each iteration, the unprotected combinations of certain size are protected by replacing a group of items by a more generalized item using a given generalization hierarchy. For each unprotected combination of items, the algorithm examines all possible generalizations and finds one that incurs the least information loss and satisfies k^m-anonymity.

Hierarchy-based generalization have also been integrated with suppression by some works [70, 12]. Liu *et al.* [70] proposed a top-down greedy method which replaces a set of public items with their generalized representations and suppresses some items to enforce k^m-anonymity. Starting with the most generalized cut, which is to generalize all the items to the root of a given taxonomy tree, the algorithm greedily attempts to find a more specialized cut, giving the required privacy protection while incurring less information loss, until no such specialization is found. To find a specialized cut, the method evaluates all possible nodes of the current cut and specializes the node causing maximum information gain due to specialization and incurring minimum suppression cost. The suppressions, required for a given generalization cut, is also determined by greedily constructing a tree in top-down manner which starts by assuming that all items are to be suppressed and iteratively removes an item from the suppression list, which is the most promising in terms of privacy and utility. For performance reasons, the algorithm works in m rounds to progressively protect the unprotected combination of items. That is, at an i_{th} round, it ensures to achieve k^i-anonymity. Cao *et al.* [12] proposed ρ-uncertainty, a model to limit the probability of inferring any non-public item, given that the attacker may know any sensitive and non-sensitive items of a record. The potential set of sensitive inferences are framed as sensitive association rules, with a set of items as antecedent assumed to constitute the background knowledge of an attacker.

They also proposed a top-down algorithm to achieve this. The method first iteratively suppresses non-public itemsets to control the confidence of the rules involving sensitive items only, and then generalizes public items to ensure that no sensitive association rule has confidence greater than ρ. The generalization starts with all non-sensitive items mapped to the most generalized item in a given hierarchy. The algorithm then iteratively specializes a generalized item causing maximum possible information gain, until no information gain can be harnessed anymore, without violating the ρ-uncertainty.

Set-based Generalization Methods

All the methods discussed in the previous section require the use of a generalization hierarchy. In such methods, if an original item is generalized to any of its ancestors, all of its siblings are forced to be generalized to that common ancestor in the hierarchy too. This limits the search space considered by these methods. Another set of methods overcomes this limitation by using set-based generalization, which is to replace a combination of original items with their set representation. Set-based generalization was first introduced by Li *et al.* [69] to anonymize transaction data, and has been shown to retain utility better, since it allows to explore a larger search space [75, 69, 73].

Loukides *et al.* [73] also showed that full-subtree recoding is a specific case of set-based generalization, where the generalizations that original items are mapped to are restricted to their ancestors in a given taxonomy tree. So the methods based on set-based generalization can also be restricted to find a solution based on the given taxonomy tree. The method proposed by Loukides *et al.* [73] is one such example. They proposed a constraint-based k-anonymity model that allows to specify privacy and utility constraints and proposed a heuristic, constraint-based anonymization of transactions (COAT), which protects transaction records by ensuring that all potentially linkable itemsets (specified by the privacy constraints) appear at least in k transaction records, using allowable generalizations (specified by the utility constraints). To achieve this, COAT iteratively performs the anonymization. During each iteration, it finds and protects an unprotected privacy constraint whose itemset specified in the privacy constraint appears most frequently in given dataset. To protect a privacy constraint, the algorithm iteratively selects an item from the itemset specified in the privacy constraint that appears least frequently and generalizes it. The generalization is made using the generalization incurring minimum information loss from all allowable generalizations, specified by the utility constraints and suppresses it if no utility constraint exists to generalize it any further. The algorithm continues generalizing or suppressing items, until the current privacy constraint is protected and then moves on to the next unprotected privacy constraint. The algorithm stops when all the privacy constraints are protected. The method does not require a hierarchy for generalization. The privacy protection is achieved by a constraint specification model which allows to specify the privacy requirements and does not make any assumption on the number or size of itemsets to be protected. However, the model anonymizes data based on the utility constraints. This may result in the anonymization solution with good data utility for intended utility requirements and may ignore the anonymization solution which gives better utility in general.

Loukides *et al.* [74] proposed PS-rules model which allows to specify the privacy constraints, and Rule-Based Anonymization of Transaction data (RBAT), a top-down specialization method which generalizes given transaction data to protect the PS-rules. As shown by Table 2.4, comparing to other methods, RBAT has some desirable features. For example, it ensures the protection against both identity and sensitive itemset disclosure attacks. The method does not require a generalization hierarchy. The PS-rules model [74, 75] allows privacy constraints to be specified, and does not make any assumption about the privacy constraints such as the itemsets of certain size must be protected. Therefore, it provides good data utility.

RBAT is also a good representative of other global anonymization methods. For example, global anonymization methods are mostly iterative and each iteration involves a number of anonymization operations to be performed in sequence. RBAT can also be used to achieve the privacy constraints specified by most of the other transaction data anonymization models such as k^m-anonymity [114], (h, k, p)-Coherence [132], complete k-anonymity [51], and ρ-uncertainty [12]. For example, specifying the privacy constraints consisting of all itemsets of size m using PS-rules and setting RBAT to achieve identity disclosure only, will allow to achieve the k^m-anonymity. We therefore consider the parallelization of RBAT in this work.

2.2 Non-Parallel Methods for Scalable Data Anonymization

The scalable anonymization of large transaction data has been considered in centralised settings. Data indexing and sampling are two commonly used techniques. Disk-based methods with external indexing is one way considered by existing methods. Iwuchukwu *et al.* [55] proposed the application of spatial indexing to k-anonymizing data. The proposed method uses a multi-dimensional R-tree. Each node in the tree represents a generalized representation. A path from the root node to a leaf node produces a set of records in a given dataset that satisfy the generalization constraints imposed by the path followed to reach the leaf node. For example, consider 3-anonymizing a relational dataset shown in Table 2.5. The possible R-tree constructed is shown in Figure 2.2.

LeFevre *et al.* [63] also proposed a technique for scaling an existing generalization method, Mondrian [64], to datasets larger than the available memory. The algorithm is based on the idea of decision tree construction method, RainForest [37]. Starting with all attribute values generalised to the root, the algorithm scans the input dataset D to collect some statistics (depending on the split criteria), and creates a frequency group. It then chooses an allowable split attribute based on this frequency group. D is scanned again to create m partitions based on the split attribute. The partitions are written to the disk, if they are larger than available memory. The process is recursively repeated

Method	Search method	Techniques	Privacy Model	Attacks	Hierarchy-based	Privacy Assumptions
RBAT [72]	Top-Down	Global Generalization	PS-rules	Both	no	User-specified Constraints
AA [114]	Bottom-up	Global Generalization	k^m-anonymity	Identity Disclosure	yes	All m-sized itemsets need to be protected
Greedy [132]	Bottom-up	Local Generalization	k^m-anonymity	Identity Disclosure	yes	All m-itemsets are to be protected
Anonymize [51]	Top-Down	Local Generalization	k-anonymity	Identity Disclosure	yes	All itemsets need to be protected
LRA [115]	Bottom-up	Local Generalization	k^m-anonymity	Identity Disclosure	yes	All m-itemsets are to be protected
mHgHs [70]	Top-Down	Global Generalization+ Suppression	k^m-anonymity	Identity Disclosure	yes	All m-itemsets are to be protected.
TDControl [12]	Top-Down	Global Generalization+ Suppression	ρ-uncertainty	Sensitive Itemset Disclosure	yes	All sensitive association rules must be protected.
COAT [73]	Iterative	Global Generalization+ Suppression	Privacy and utility specification	Identity Disclosure	no	utility requirements specification

Table 2.4: Comparision of transaction data anonymization methods

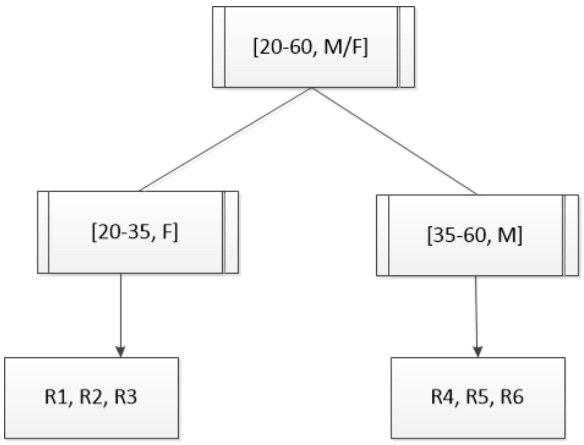


Figure 2.2: A example R-tree for k-anonymous data

ID	**Age**	Sex
1	21	F
2	22	F
3	35	F
4	36	M
5	45	M
6	55	M

Table 2.5: An example Dataset T

in depth-first manner, until no split is possible. Using such disk-based approaches with high-speed disks may address the problem of anonymizing data larger than main memory but these disk-based methods in general limit the performance of the method, to the capabilities of current hardware technologies such as disk I/O speed. Also, the use of resources will still be limited to the processing and storage capability of a single machine. For example, accessing a large amount of data from disks frequently may lead to a performance bottleneck. Therefore, dealing with increasing scale of data using disk-based methods may not be promising in terms of scalability.

The use of sampling techniques have also been considered by some works [63, 65, 75]. Lefevre *et al.* [63] proposed the use of random sampling for scalable k-anonymization

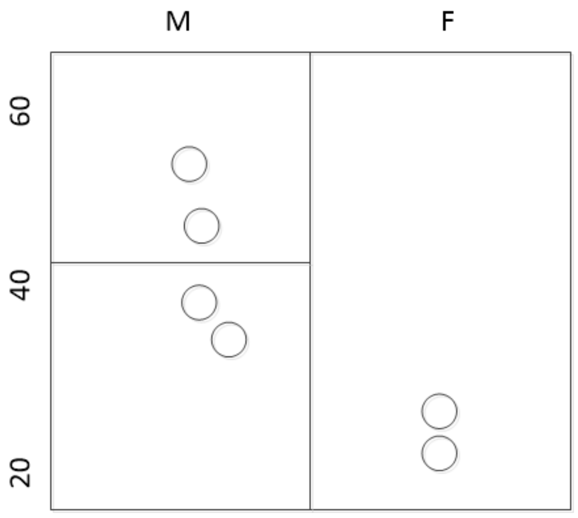


Figure 2.3: A example 2-anonymous data using Mondrian

of relational data. The method scans the input data and generates a random sample that fits in the available memory and applies the Mondrian method to anonymize the data. The sample is used to construct the partition tree by choosing the allowable splits. For example, consider a sample of six records used to create a partition tree as shown in Figure 2.3. The data sample is first partitioned vertically across the sex attribute and then horizontally across the age attribute. This creates three different partitions satisfying 2-anonymity. The partition tree created by the sample is used to anonymize original data and any splits violating the privacy constraints are undone. Most closely related to our work is the sampling-based method proposed by Loukides *et al.* [75] to anonymize transaction data. The algorithm uses top-down specialization. It selects a random sample of pre-determined size. Starting with all the items mapped to the most generalized item, it recursively performs specializations until no specialization can be made without violating any privacy constraint. Each time, a specialization operation is performed, the privacy constraints are checked using the sample. The set of generalizations acquired using the sample are then revised using the top-down and bottom-up cut-revision phases. The top-down revision phase further attempts to specialize the generalized items in order to find a solution with same privacy level but better utility. The bottom up cut-revision phase ensures the privacy protection by generalizing the items further, if the anonymized solution found using the sample does not give the required privacy protection for the whole dataset. Although the above approaches solve the problem of the limitations put by disk-based methods, the privacy and utility achieved by the anonymized solution using the sample may not guarantee the same level of privacy and utility for the original data. Therefore, the anonymized solution may need to be revised using the whole dataset.

2.3 Possible Choices for Parallelization

There are two important aspects of a parallel system: an algorithm for carrying out the computation in parallel and the parallel model used for its design. The choice of a parallel model affects the way the parallel algorithms are designed and help determines their different characteristics such as parallelism overhead, degree of parallelism *etc*. According to the existing work [104, 106, 11], a parallel model must be general enough to model a range of physical architectures. This makes the parallel solutions portable to various architectures. The cost of the algorithms estimated using a parallel model must not differ too much from its cost in practice on the targeted physical architectures. The model must also be manageable. That is, it must abstract the details such as parallelism, communication, synchronization *etc.*

Several parallel architectures exist [11, 66, 130] supported by different parallel models. Based on characteristics such as the instructions and the data processing patterns, Flyyn [32] broadly classified these architectures into four categories: SISD (Single Instruction, Single Data Stream), SIMD (Single Instruction, Multiple Data Stream), MISD (Multiple Instruction, Single Data Stream) and MIMD (Multiple Instruction, Multiple Data Stream). Two important families of these parallel architectures suitable to our problem of large-scale data processing are SIMD and MIMD. For example, a cluster of workstations or massive parallel processor allows the use of multiple processors to perform large-scale computations in parallel. Below, we survey some important parallel models supported by these classes of physical architectures and analyse their characteristics.

2.3.1 PRAM

Fortune and Wyllie [33] proposed PRAM (Parallel Random Access Machine), a parallel extension of the random access machine model. It consists of an unbounded number of processors, each with its own local set of registers. All the processors are connected to an unbounded global memory, shared among the processors via which they communicate. A PRAM processing cycle consists of a number of time steps. During a unit time step, each processor may read any data from global memory into its local register set, perform any computations over the data stored in its local registers and write any data to the shared memory.

Although the model is simple to adopt and had led itself to be a widely accepted research tool, as evidenced by a number of solutions designed based on it [59], the model is based on some unrealistic assumptions and neglects some practical issues. For example, the cost of a PRAM algorithm assumes that cost incurred by a processor to access its local registers and the cost to perform inter-processor communication is the same. All processors are also assumed to work synchronously. Therefore, the performance of the parallel solutions in practice may be far worse than estimated at the time of their design [41, 20].

2.3.2 BSP

Valient [121] proposed the BSP (Barrier Synchronous Parallel) model, in an attempt to overcome the unrealistic assumptions made by PRAM about the physical characteristics of the underlying architecture. The model assumes the use of two-level memory: local and global. It also differentiates the costs associated with accessing local memory and global memory. A BSP system consists of a number of components, each of which is assumed to have a processor and a local memory associated with it. All the components are assumed to be connected to all other processors via some means of point-to-point communication.

The computation consists of a sequence of steps called supersteps. During a superstep, each component performs its assigned tasks locally and any necessary communication with other components. Every l time steps, a global check is made to determine if the current superstep is completed by all the participating components. If all the tasks of current superstep is finished, the components are synchronized and moved to the next superstep. Otherwise, the system waits for another l time steps to allow the completion of unfinished tasks of the current superstep.

The model takes into account the different characteristics related to parallel design and the physical architecture used such as computational cost, communication and synchronization cost, bandwidth inefficiency *etc.*, so the estimated performance of the parallel method on any given architectural settings is more predictable than PRAM.

The parallel programs written in different implementations of BSP are portable to various architectures [105, 45]. However, the implementations of the model still require specifications related to parallel execution such as synchronization etc.

2.3.3 MapReduce

Dean *et al.* [22] proposed MapReduce for large-scale data processing in parallel. It simplifies the design of the scalable solutions for processing a large amount of data, by hiding lower-level details of parallel execution such as fault-tolerance, load balancing and synchronization.

MapReduce follows the client/server architecture. One of the machines works as a master node which performs scheduling of tasks, coordinates the distribution of data to the worker nodes and holds, all the book-keeping about the workers, jobs and the data.

Each client node is assumed to have its own local memory and a global distributed memory shared among all nodes.

Working Mechanism: A MapReduce computation consists of a number of MapReduce rounds. Each MapReduce round consists of two computational stages called Map and Reduce, intermediated by a communication stage called Shuffle. An schematic representation of a typical MapReduce round is shown in Figure 2.4.

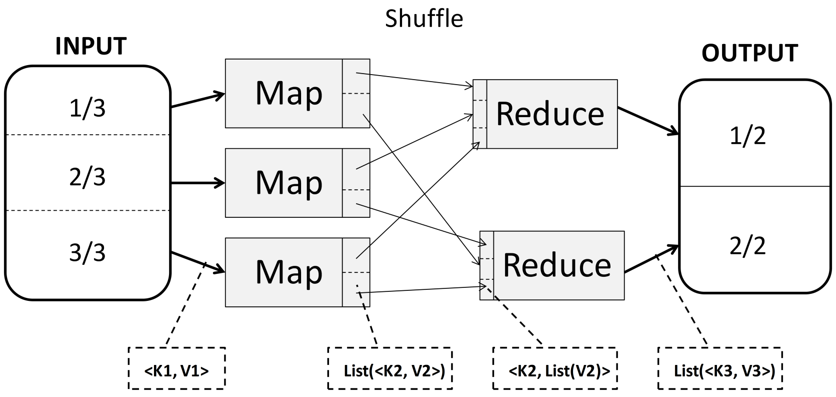


Figure 2.4: A schematic representation of a MapReduce round

The map/reduce computations take input and produce output as a set of key/value pairs. The input and output are usually stored in a distributed file system [38]. A typical MapReduce cluster consists of a single master node (computer) which schedules and monitors different map/reduce data processing tasks over the slave nodes. Conceptually, a MapReduce round consists of map, shuffle and reduce phases, and can be expressed as:

$$Map(k_1, v_1) \rightarrow list(k_2, v_2)$$

$$Reduce(k_2, list(v_2)) \rightarrow list(k_3, v_3)$$

In the Map stage, the given data are logically partitioned into a number of disjoint subsets and each subset is assigned to a mapper. The assignment of partitions to mappers is made dynamically and is determined at runtime. A mapper (in parallel to other mappers) reads a record in the form (k_1, v_1) from its assigned data partition, performs the user-specified computation, and outputs another set $list(k_2, v_2)$. If a mapper runs out of memory at this stage, the part of output is temporarily moved to the local file system (LFS).

The output pairs produced by mappers (intermediate output) go through a shuffle phase. At this stage, each mapper locally performs the partitioning of its output. Given r reducers, the key space of intermediate output is divided into r partitions in a way each partition has almost an equal number of distinct keys. This is to ensure that the amount of work associated with each partition is approximately the same. The partitioning is usually done via a hash function h on a key k of each pair, *i.e.* $h(k)\ mod\ r$. This ensures that the pairs with the same key from different mappers are assigned to a single reducer. The partitioning strategy is based on the assumption that the amount of work associated with each key of intermediate output is the same. Note that a mapper output may not be local to its assigned reducer. In this case, shuffle also involves transferring mapper output to a reducer over the network. The shuffle phase starts as soon as the first mapper finishes its processing, so it may overlap with the map phase.

A reducer r_i acquires its assigned part of map output by copying the pairs (k_2, v_2) from $b_{i,j}$, an i_{th} partition from the j_{th} mapper ($1 \leq j \leq M$), sorts all the assigned pairs by their keys to construct $(k_2, list(v_2))$ and processes $list(v_2)$ for each distinct key k_2 using a user-specified reduce function. The output produced by the reduce phase can be the final output or can be input to another mapper in the next MapReduce round.

It is worth noting that there is an overhead for achieving parallelization in MapReduce. This mostly consists of the cost of initializing a MapReduce job, I/O cost of reading input and writing output by mappers and reducers during the map and reduce stages, and the cost of shuffling mapper output over the network to reducers. It is also worth observing that output produced by mappers is an input to reducers, which is transferred via shuffle phase. Therefore, less the intermediate output produced by the map stage, the lower the cost of transferring it over the network will be and the lower the I/O cost of writing intermediate output and reading input by the mappers and reducers will be. The number of mappers and reducers must also be specified based on the available resources. Setting them to a high value may result in a small amount of work for each mapper or reducer, and the performance gain by parallel processing may be offset by their setup cost. Setting them to a low value on the other hand, may result in ineffective resource utilization.

Example 1. *Consider a simple word count using MapReduce. A given text is divided into $M = 3$ chunks and assigned to mappers. During the map stage, a mapper is assigned a chunk of text. Each i_{th} line of the assigned chunk is read in the form of a pair $\langle i, l_i \rangle$, where l_i represents the contents of i_{th} line in the partition. It then counts the occurrences of each distinct word in l_i, and outputs $\langle word, occurrences_j \rangle$ for each distinct word. Once the lines are processed, the output pairs of each mapper are divided into $r = 2$ partitions. For each map output pair $\langle word, occurrences_j \rangle$, it computes $h(word)$ mod r to decide to which of the two reducers the pair is sent to. The pairs are then copied to their assigned reducers. A reducer sums up all local $occurrences_j$ ($1 \leq j \leq M$) of a $word$, and outputs the global count $\langle word, occurence \rangle$.*

MapReduce and other Parallel Architectures: Mapreduce was originally designed to run on a large cluster of commodity machines, but later was also adopted to other parallel architectures. Some existing works [49, 108, 16] proposed a design and implementation of MapReduce over graphics processing unit (GPU) clusters. Chen *et al.* [17] attempted to scale up MapReduce applications by proposing a number of work partitioning schemes which leverages both CPUs and GPUs to perform the given computations in parallel. Karloff *et al.* [58] and Goodrich *et al.* [44] showed that MapReduce can also be used to simulate PRAM and BSP models.

Unlike other parallel models such as BSP, MapReduce was not introduced with any formal cost estimation formulation. Therefore, an analytical study taking into account the characteristics of physical architecture and the given problem instance is needed to help predict the cost of a parallel solution.

2.4 MapReduce and Privacy Protection

The privacy protection using MapReduce has recently been considered. Some methods allow the computations to be performed using MapReduce while controlling the access to original data or the sensitive information therein. Roy *et al.* [102] proposed Airavat, a MapReduce-based system that ensures security and privacy guarantees, when the distributed computations are performed over sensitive data stored in a cloud. The system prevents the information leakage which might occur due to access available to original data or output of the computation performed by the untrusted data users. The privacy and security is achieved by enforcing access control policies over the data access via system resources such as network connections or storage channels and by also ensuring that the output of the computations are differentially private [26]. Zhang *et al.* [136] studied the problem of performing MapReduce-based computations over hybrid clouds. They adopted MapReduce to perform secure computations using the non-sensitive data in public clouds and the sensitive data in private clouds. They proposed Sediac, a MapReduce-based system which partitions a computing job according to the required privacy level. The tasks performing computations over public data are scheduled on a public cloud and those processing private data always stay on a private cloud. A two-stage process is used to aggregate the results. In the first stage, the public and private clouds individually perform their local aggregation. The aggregated results from public clouds are then brought to private cloud for final aggregation. The use of cryptographic methods to perform information-retrieval operations in a privacy-preserving way using MapReduce has also been considered. Blass *et al.* [7] proposed a MapReduce-based privacy-preserving method to search words from the data stored in public clouds in encrypted form. During the map stage, each node performs the search over encrypted data using an instance of private information retrieval method on its assigned data partition. The intermediate search results are aggregated by the reduce phase. Mayberry *et al.* [78] proposed a MapReduce-based solution to the problem of protecting the privacy of cloud users from untrusted cloud providers. More specifically, they proposed a method to protect the information leakage about the data access patterns of cloud users. Given n files stored in a cloud, their MapReduce-based implementation of a private information retrieval protocol allows an encrypted query to be performed over distributed data in a way that the information about which files were accessed is not disclosed to the cloud provider. All these privacy-preserving methods focus on the privacy and security concerning data sharing over clouds rather than focusing on achieving scalability. These methods also make different privacy assumptions and can not be adopted to our scenario. For example, adopting the access control method will need to limit the access of data to trusted users and is orthogonal to our scenario of releasing data to untrusted recipients.

Leveraging MapReduce to address scalability issues in privacy-preserving methods has also recently been investigated. Zhang *et al.* [137] proposed a MapReduce-based local recoding method to k-anonymize the data to prevent proximity privacy attacks [127], which happens when the sensitive values in an anonymized group are not diverse enough. The proposed method uses clustering and consists of two phases. The first phase uses a method similar to Lloyd's k-means clustering [71] and partitions dataset into a pre-defined number of partitions so that records with similar non-sensitive attribute values and contain semantically distant sensitive values are placed in the same partition. During the second stage, each partition is assigned to a worker node, which then uses complete linkage agglomerative clustering to produce groups of records, each of size at least k. The non-sensitive attribute values in each group are replaced by their most generalized representation in the group. The groups from all the worker nodes are finally combined to form the anonymized dataset. Since the partitioning is made based on the data distribution, the created partitions may not be balanced in the associated workload.

More closely related to our work are the MapReduce-based top-down [139] and bottom-up [138] methods to achieve k-anonymity. The top-down specialization method proposed by Zhang *et al.* [139] partitions data into a number of partitions by randomly assigning records to the partitions. All partitions are then anonymized in parallel. Each partition is anonymized by a number of worker nodes using a MapReduce-based top-down specialization job. To ensure that all the data partitions achieve the same level of anonymization, the algorithm merges all the generalization cuts into a single most generalized cut, by choosing the most generalized representation of an attribute value used in any of the data partitions. Zhang *et al.* [138] proposed a MapReduce-based implementation of bottom-up generalization method. Starting from the original attribute values, the algorithm iteratively finds the best generalization and updates the anonymized dataset accordingly, until all the generalizations occur in no less than k records. To find the best generalization candidate, the algorithm scans the dataset and collects the required statistics in a single MapReduce round. Similar to our method, the algorithm performs all the computations requiring full data scan in parallel and keeps the updated anonymization level which is used to construct the anonymized dataset in the end using single MapReduce round. But this may lead to poor performance, as evidenced by our preliminary experiments (Section 4.4). Also, all the methods are for relational data so do not take into account the characteristics of transaction data. Furthermore, these method also achieve k-anonymity. Unlike k-anonymity however, the models for protecting sensitive information disclosure do not have the property of monotonicity. That is, achieving the required protection on individual data partitions may not guarantee the same or higher protection level on the overall dataset. So such methods can not trivially be extended to achieve the protection against both types of disclosure.

2.5 Other applications of MapReduce

MapReduce has been adopted to address scalable data processing in various applications [67, 99, 94, 29]. Despite its wide-spread adoption, its simplicity introduces some limitations in the way the processing must be performed. This often causes degraded performance to an unacceptable level when performing various processing tasks [23]. In the following, we highlight some of these limitation and describe possible solutions in the light of existing applications.

Load balancing: In order to achieve the effective resource utilization, workload balance among processing nodes is important. In MapReduce, the parallel tasks are performed at both the map and reduce stages. These tasks are created by partitioning input data at the map and reduce stages respectively. Therefore, workload balancing must be done at both stages.

Existing methods employ different load balancing strategies. Okcan *et al.* [89] proposed a randomized method to map an arbitrary join condition to MapReduce with the goal of balancing workload among reducers. Given two datasets S and T to join based on an arbitrary condition θ, the algorithm first constructs a $|S| \times |T|$ matrix, with an entry $M_{i,j}$ set to true if the i_{th} record ($1 \leq i \leq |S|$) of S and j_{th} record ($1 \leq j \leq |T|$) of T satisfies θ condition. The records corresponding to the entries satisfying the condition are then mapped to reducers in a way that all reducers get an equal number of records. Finally, the reducers generate output pairs of the joined records. Vernica *et al.* [123] studied the problem of finding similar pairs of records from a dataset using MapReduce. The method first reads the input records in parallel and produces a list of tokens (words) that appear in the join-attribute value ordered in ascending order of their frequency. The map stage of the second stage orders the tokens of each record in ascending order of their global frequency using the frequency list created in the previous stage and creates a prefix of size k. For example, consider an ordered record $\{call, back, I, will\}$ with the token $call$ appearing least frequently in the given dataset. The prefix of size 2 is $\{call, back\}$. The records with at least one common token in their prefixes are sent to the same reducer to perform the join operation. Using the least frequent tokens of records as their prefixes will eliminate the skew which could have arisen due to the presence of frequent tokens. This results in workload balance among reducers.

Communication among processing nodes: MapReduce provides limited communication among processing nodes. At each MapReduce round, the processing nodes are not allowed to communicate with each other during the map and reduce stages except at the shuffle stage. That is, when a processing node finishes its assigned map tasks, the map output is sent to the reduce tasks for further processing.

The lack of communication among the nodes may cause a large amount of data to be shuffled to the nodes and hence may result in high shuffle cost. Existing works attempted to reduce the communication cost by collecting some statistics from given data and creating independent data partitions based on such collected statistics. Li *et al.* [67] proposed PFP, an adaptation of the FP-Growth method for association rule mining [48]. They proposed a method to partition a given dataset into a number of subsets so that each subset can be processed independently and the amount of data to be communicated among the processing nodes is minimized. The algorithm first computes the support of all distinct items of the given dataset D to find the set of frequent items. At this stage, the algorithm also discovers the universe of items I. I is then divided into G groups using a single machine. The list of all the groups and a data partition D_j is assigned to a mapper. The mapper then creates a pair containing a group identifier and a list of all the dependent records in D_j for each group. At the reduce stage, a reducer is assigned a group and receives all dependent transactions from the map stage. It then builds the local FP-tree and the conditional FP-tree recursively by computing the frequent itemsets. The top-K most supported locally found patterns are emitted by each reducer and are used to construct a list of globally frequent itemsets. However, creating the independent data partitions may also require to replicate some data to more than one node which may increase the memory requirement on each processing node. Data elimination is another way to reduce the communication cost. Park *et al.* [96] proposed a MapReduce-based method to perform skyline processing from large data volumes. The method first uses a data sample to construct a histogram called sky-quadtree. The leaf nodes of the quadtree with non-skyline sample points only are marked as pruned. The unpruned regions are used to partition the whole dataset into regions and each region is mapped to a processing node for local skyline computation. The local computation results are sent to a single machine for global skyline output.

Another challenge is to perform global computations. For example, finding a maximum value globally from locally found maximum values may require to perform a part of the computation serially. This limits parallelism. Mullesgaard *et al.* [83] proposed a method to compute global skyline results without limiting the number of processing nodes to one only. Their method divides the given data space into parts. The partitions are mapped to different slave machines for local skyline computation. The locally found skyline results are then grouped together in a way that each record and its possible set of dominating records are grouped together. The groups are mapped to different processing nodes and each processing node computes a part of the global skyline.

Iterative processing: In MapReduce, each round is considered as separate. That is, no data is kept between two MapReduce rounds. Setting up each MapReduce round incurs parallelism overhead including the cost of setting up tasks on processing nodes and reading data associated with the tasks. Algorithms designed to employ a large number of MapReduce rounds may not be scalable and efficient, due to large parallelism overhead. Recent algorithmic formalizations of MapReduce have also focused primarily on optimizing the number of rounds used to solve a problem [58, 30]. Unfortunately many applications exist where the number of required iterations is a function of some characteristics of the given input such as size or distribution of certain values, and they are not directly supported by the framework and can cause considerable performance overheads. Some examples of such applications are K-means [77], deterministic annealing clustering [101], and Pagerank [8].

One way to address this problem is to modify the MapReduce paradigm. Some existing variants [54, 133] generalize the data flow model supported by MapReduce. For example, Dryad [54] proposed computation to be specified as a directed multigraph with nodes representing the processes and the edges representing the communication channels among them. Map-Reduce-Merge [133] extends the existing map-reduce by adding the merge step which can merge the output of two different MapReduce rounds without setting up an extra MapReduce round. These generalized frameworks support more complicated data flow patterns, but still require computations to be specified as a directed acyclic graph.

Some other existing works [9, 27, 135] focus on addressing the problem of efficiently supporting iterative methods, and proposed some other variations of the framework. Ekanayake et al. [27] and Bu et al. [9] proposed to avoid reading unnecessary data repeatedly from distributed storage, by identifying invariant data and keeping them locally over iterations. Twister [27] also attempted to reduce parallelism overhead by initializing mappers and reducers once in the beginning and pushing the map outputs directly over the network to reducers instead of materializing them locally. Haloop [9] caches and indexes the invariant reduce input and output across all reducers, in order to reduce the shuffling cost and the cost of setting up a separate MapReduce job to evaluate the termination condition. Spark [135] allows to keep the data partitions in the memory of the worker nodes across different iterations, and also facilitates the use of shared variables to distribute the read-only data and accumulate the information from worker nodes.

However, these variations assume that most of the data remain static between iterations. Iterations of some methods such as RBAT do not satisfy this requirement. Furthermore, these variations also limit some features of the standard MapReduce framework. For example, forcing the data to remain locally on fixed nodes means that the tasks involving such data cannot be scheduled to be processed on multiple computing nodes which may result in poor performance of the applications, especially over heterogeneous clusters. Also, some variations of MapReduce such as Spark [135] only supports reduction of map output locally in the worker nodes (analogous to combiner in MapReduce), but final reduction of all map output is only allowed to be done by a single node.

This means that not all the resources are utilized at the reduce phase. Also, this may not be scalable when the output of the map stage can not fit into the memory of a single node.

Existing methods also attempted to use design-side strategies to reduce the number of iterations. Ene *et al.* [29] used MapReduce for k-median and k-center clustering of large datasets. The algorithm takes a constant number of MapReduce rounds to construct a well-represented sample from the given dataset. At each round, the points not represented by the sample are read in parallel, and the sample size is increased by adding the points with a certain probability. All the sampled points, along with their pairwise distances and any additional statistics from the given dataset (which is also computed in parallel), are mapped to a single machine to run an instance of the sequential k-median or k-center clustering method. Riondato *et al.* [99] proposed PARMA, a MapReduce-based approximation method for association rule mining. The algorithm takes two MapReduce rounds. During the first MapReduce round, the given dataset is scanned to create N random samples, which are mined independently by the reducers, each using an instance of a sequential pattern mining method. The results created by the samples in the previous phase are aggregated to obtain an ϵ-approximation of the exact output with the probability $1 - \delta$, where ϵ and δ are pre-defined parameters. The use of sampling may reduce the number of MapReduce rounds employed, but this may be at the cost of some loss of quality or accuracy in the output. Panda *et al.* [93] adopted MapReduce to construct a part of the regression tree from large training data. The algorithm is based on an existing top-down method [24]. At the root node of the tree, the algorithm requires examining the entire training dataset, in order to find the best split predicate which is used to partition the data. The process is recursively repeated until the stop criteria is met. The tree constructed so far is maintained in a distributed cache and the nodes to be considered for further splitting are maintained in two queues held in the memory of master node. One of the queues holds the nodes to be split sequentially. The splits of such nodes are performed in the memory of a single machine. The second queue holds the remaining nodes. The algorithm reads the entire training dataset in parallel, and finds a set of good splits for these nodes. For each node, the locally chosen set of splits, along with some statistics is collected centrally on single machine, which aggregates the computations and outputs the best split.

Despite the focus on the problem of iterative computations (as evidenced by the work surveyed here), scalability to large datasets in some iterative applications can still be achieved by direct implementation of the sequential methods in MapReduce. These methods keep the processing logic from their sequential counterpart. Verma *et al.* [122] proposed a MapReduce-based method to make the selecto-recombinative genetic algorithms [43] scalable to large datasets. Starting with initializing a population with random individuals in a single MapReduce round, the algorithm iteratively creates new populations by selecting and combining a set of individuals from the current population, until some convergence criteria is met. Each iteration is performed in a single MapReduce round. During the map stage, the mappers compute the fitness value of individuals in the current population, and shuffle the output to a randomly selected reducer. A reducer (in parallel with other reducers) creates new individuals from a given set of individuals by sections-and-recombination. Papadimitriou *et al.* [94] proposed DISCO, an iterative distributed co-clustering method using MapReduce. The algorithm alternatively performs row or column-wise scan of the given matrix, assigning each row (or column) a group label to minimize the overall error. Each scan is performed using a single MapReduce round. During the map stage, a mapper reads a row, assigns a group label, and computes the associated per column statistics for that row. The rows with the same group label are sent to the same reducer which merges them into a single cluster and the associated statistics is aggregated for each group. Each iteration is followed by a global synchronization of the group matrix and a column-wise data scan, which is also performed in the same way as row-wise scan, but on the transposed matrix. This continues until the error does not decrease between two consecutive iterations. McNabb *et al.* [79] proposed MRPSO, a MapReduce-based implementation of particle swarm optimization. Starting with initializing a set of particles (points) with some random positions and velocities, the algorithm iteratively attempts to find the best possible position and velocity of each particle, with the goal to optimize some given function. At each iteration, the mapper takes individual particles, computes their new position and velocity, and also evaluates the input function to ensure that its local best so far is updated. The updated state for the particle is shuffled to all its dependent particles, which is then used by the reducers to update the global best in the neighbourhood of each particle. These methods highlight the fact that MapReduce can be used for scalable computation from large datasets without sacrificing the quality or accuracy of output results.

2.6 Summary

In this chapter, we reviewed the existing work relevant to our study of large transaction data anonymization. We surveyed different models and privacy techniques used for data anonymization. We discussed different generalization-based methods for transaction data anonymization and compared them in terms of their ability to offer privacy protection and retain data utility. We have seen that set-based global generalization methods prevent one or both types of disclosures without significantly compromising data utility. But all these methods search the solution space in an iterative manner. We also surveyed the existing non-parallel attempts to address the problem of scalable data anonymization. We have seen that techniques such as sampling or indexing are not sufficient to deal with increasing data volumes and may compromise the privacy and utility of anonymized data.

We also have studied different parallel models and discussed their features. We have seen that MapReduce provides certain features such as abstraction to parallel execution details and has been applied to address the problem of scalability in various domains. Some recent studies have also focused on the scalability issue of privacy-preserving methods but the problem of scalable transaction data anonymization still need to addressed.

We also surveyed other applications of MapReduce for scalable computations and studied some limitations of the paradigm. We have seen that MapReduce allows limited communication among processing nodes. Performing iterative computations may incur large parallelism overhead and have been addressed in different ways. We have seen that adopting modifications of MapReduce to address these limitation will limit certain features provided by the framework. Therefore, these must be considered when designing parallel solutions using MapReduce. In the next chapter, we study how MapReduce can be used to parallelize RBAT, a set-based generalization method and discuss different issues to be considered for our parallel solution.

Chapter 3

RBAT and MapReduce

In this chapter, we study how MapReduce can be used to make set-based generalization methods scalable to large data volumes without affecting the level of privacy and utility they offer. More specifically, we formulate a MapReduce parallelization of RBAT. Section 3.1 provides some important notations and concepts, and formally defines our problem. We give an overview of RBAT in Section 3.2. Section **??** presents a brief overview of how MapReduce works. Finally, Section 3.3 gives a general framework to consider possible ways of parallelizing RBAT in MapReduce, and discusses the important issues around these possible solutions.

3.1 Preliminaries and Problem Definition

Let $I = \{i_1, \cdots, i_{|I|}\}$ be a finite set of literals called items. Let $|I|$ be the size of I. A transaction $t = \langle i_m, \ldots, i_n \rangle$ is a record where each i_j ($1 \leq j \leq |I|$) is a distinct item of I. A transaction dataset $D = \{t_1, \ldots, t_{|D|}\}$ is a collection of transaction records over I. Any subset $\lambda \subset I$ is called an itemset over I. The frequency of an itemset is called its support in a given dataset.

Definition 1. *(Support) Given an itemset λ, its supporting transactions in D are the transactions containing all items of λ and the number of such transactions, denoted by*

$\sigma_D(\lambda)$, *is called its support. i.e.* $\sigma_D(\lambda) = |\{t \in D \wedge \lambda \subset t\}|$.

Table 3.1 shows an example transaction dataset. $\langle b, c, e, \mathbf{g}, \mathbf{h} \rangle$ is an example transaction, where boldface indicates sensitive items. (b, c) is an itemset, and its support $\sigma_D(b, c) = 1$ with the first transaction being its supporting transaction.

Patient	*Diagnosis Codes*
Alice	b, c, e, **g**, **h**
Bob	a, c, d, **i**, **j**
Tom	a, f, **l**
Jim	b, e, **g**, **h**
Jerry	d, f, **l**

Table 3.1: An example dataset D

We partition I into two disjoint subsets P and S such that $P \cup S = I$ and $P \cap S = \varnothing$, and represent their size by $|P|$ and $|S|$ respectively. S contains items that represent the sensitive information about the associated individuals who need to be protected, and P contains all other items called public items. We assume that S needs to be published intact, since it is often required by applications [132] and that an attacker may have knowledge about individuals in the form P. Also, each transaction record is assumed to contain one or more items from P and as well as from S.

The background knowledge of an adversary is considered to be in the form of an itemset. When the itemset has support below a certain threshold k then the adversary may be able to use the itemset to associate an individual to his transaction with the probability higher than $\frac{1}{k}$, thereby breaching privacy. For example, consider the dataset D shown in Table 3.1. Assuming that D is de-identified (*i.e.* not containing patients' names). If an adversary knows that Bob was diagnosed with a and c, he will be able to uniquely associate Bob with his transaction, and hence can infer that Bob was also diagnosed with **i** and **j**. To protect this, PS-rules must be specified [72].

Definition 2. *(PS-rule) Given two itemsets $p \subseteq P$ and $s \subseteq S$, a PS-rule is an implication of the form $p \rightarrow s$.*

Each PS-rule captures an association between a public and a sensitive itemset. The antecedent and consequent of each rule can consist of any public and sensitive items respectively and many PS-rules can be specified by data publishers to capture detailed privacy requirements. A published transaction dataset is deemed to be protected if the specified PS-rules are protected.

Definition 3. *(Protection of PS-rule) Given a dataset D, the parameters $k \in [2, |D|]$ and $c \in [0, 1]$, a PS-rule $p \rightarrow s$ is protected in D if 1) $\sigma_D(p) \geq k$ and 2) $Conf(p \rightarrow s) \leq c$ where $Conf$ is defined as $\frac{\sigma_D(p \cup s)}{\sigma_D(p)}$.*

Condition 1 protects data against identity disclosure by ensuring that the probability of associating an individual to his or her transaction in D using the antecedent of any PS-rule is no more than $1/k$. Condition 2 prevents sensitive item disclosure by ensuring that the probability of linking an individual to a set of sensitive items specified by the consequent of a PS-rule is at most c, given that the probability of associating an individual to his or her transaction using the rule's antecedent is no more than $1/k$.

Diagnosis Codes
(a,b,c,d,e,f), **g**, **h**
(a,b,c,d,e,f), **i**, **j**
(a,b,c,d,e,f), **l**
(a,b,c,d,e,f), **g**, **h**
(a,b,c,d,e,f), **l**

Table 3.2: The most generalized dataset $\tilde{D}$

Given a set of transactions D and a set of PS-rules Θ, if any rule in Θ is not protected, then D must be sanitized. We sanitize data, using a set-based generalization approach.

The set-based generalization maps a combination of original items to a itemset consisting of all these items.

Definition 4. *(Set-based Generalization) Let $\tilde{P}$ be a partition of P. Set-based generalization is a function $\phi : P \to \tilde{P}$ which maps each item $i \in P$ to a generalized item $\tilde{i} \in \tilde{P}$ containing i.*

Note that using set-based generalization, an item can not be mapped to an empty set. Hence, each item $i \in P$ is generalized to $(i_1, \cdots, i_n)$ $(1 \leq n \leq |P|)$. We use the brackets to represent a generalized item. When $i = \phi(i)$, the item is mapped to its original representation. In such cases, we simply drop the brackets.

Set-based generalization retains data utility better than other generalization methods [87, 73, 75] due to their ability to represent a large number of possible generalizations.

Example 2. *Suppose that we require $k = 3$ and $c = 0.6$. PS-rule $ac \to h$ is not protected in D as ac has a support of 1 only. But $ac \to h$ is protected in $\tilde{D}$ given in Table 3.2 where all public items $\{a, b, c, d, e, f\}$ are mapped to (a, b, c, d, e, f) following the generalization, since ac is now supported by all transactions records and $Conf(ac \to h) = 0.5$.*

Privacy preservation is one aspect of anonymization. The other aspect is to retain the utility of data. There can be many possible generalizations of a dataset, offering the same level of privacy protection but the one which has minimum loss of information is preferred. Different utility measures have been proposed to capture the loss of information incurred by generalization. Some of them assume that the usage of published data is known at the time of anonymization. For example, multiple-level mining loss measure expresses the information loss in the detection of multi-level frequent itemsets [115], when mining is performed on an anonymized dataset instead of the original one. Classification Metric [57] assumes that the purpose of published data is to train a classifier. Other general-purpose utility loss measures have also been proposed. Normalized Certainty Penalty [131] penalizes the generalized data based on the number of its descendant nodes in a given generalization hierarchy. Generalized loss measure [56] computes the information loss as the sum of the ratio of the number of leaf nodes under a sub-tree rooted at each generalized node, to the total number of leaf nodes in the generalization hierarchy. RBAT uses the Utility Loss (UL) measure [73], which can be applied in the absence of a generalized hierarchy.

Definition 5. *(Utility Loss) Given a generalized dataset $\tilde{D}$, the utility loss of a single generalized item $\tilde{i}$ is given by $UL(\tilde{i}) = \frac{2^{|\tilde{i}|}-1}{2^{|P|}-1} \times w(\tilde{i}) \times \sigma_{\tilde{D}}(\tilde{i})$. The utility loss of the whole dataset $\tilde{D}$ is calculated as $UL(\tilde{D}) = \sum_{\forall \tilde{i} \in \tilde{P}} UL(\tilde{i})$.*

The UL measure given in Definition 5 captures the loss of information in terms of the size of the generalized itemset, its significance (weight) and its support in $\tilde{D}$. The more items are generalized together, the more uncertain we are about its original representation, hence more utility loss. $w(\tilde{i})$ assigns some penalty based on the importance of the items in $\tilde{i}$. The support of the generalized item also affects the utility of anonymized data. The more frequent the generalized item is, the more distortion to the whole dataset will be.

3.1.1 Problem Statement

Privacy-preservation has two goals: achieving a required level of protection and retaining the data utility as much as possible. More formally, given a transaction dataset D with a set P of public items, a set of PS-rules Θ, and the support and confidence parameters k and c, find the generalization $\tilde{P}$ of all the items in P using set-based generalization so that the anonymized form $\tilde{D}$ of D, constructed by replacing P with $\tilde{P}$ in D satisfies the following condition:

1. Each PS-rule of Θ is protected in $\tilde{D}$
2. $UL(\tilde{D}) - UL(D)$ is minimum, where $UL(\tilde{D})$ and $UL(D)$ are calculated according to Definition 5.

Algorithm 3.1: RBAT (D, Θ, $\tilde{i}$, k,c)

Input: Original dataset D, set of PS-rules Θ, minimum support k and maximum confidence c.

Output: Anonymized dataset $\tilde{D}$

```
1: $Q$.enqueue($\tilde{i}$)
2: while $|Q| > 0$ do
3:    $\tilde{i} \leftarrow Q$.dequeue()
4:    $\{i_l, i_r\} \leftarrow$ Split($\tilde{i}$)
5:    $D' \leftarrow$ Update($\tilde{i}_l$, $\tilde{i}_r$, $\tilde{i}$, $\tilde{D}$, $D$)
6:    if Check($D'$, $\Theta$, $k$,$c$) = true then
7:       $Q$.enqueue($\tilde{i}_l$)
8:       $Q$.enqueue($\tilde{i}_r$)
9:       $\tilde{D} \leftarrow D'$
10: return $\tilde{D}$
```

Loukides *et al.* [75] showed that finding an optimal anonymized solution, satisfying the above conditions is NP-hard. Also, the solution may not exist in some cases. For example, if there exists a rule $P \rightarrow S \in \Theta$ for which $\sigma_D(P \cup S) > |D| \times c$. We consider how to achieve scalability, while maintaining the required level of privacy and minimum possible loss of utility.

3.2 Overview of RBAT

RBAT [72] is a heuristic method for anonymizing transaction data using set-based generalization. It allows detailed privacy constraints to be specified as a set of PS-rules. The pseudocode of RBAT is given in Algorithm 3.1.

RBAT is iterative and works in a top-down fashion. Starting with all public items mapped to a single most generalized item $\tilde{i}$ and $\tilde{D}$ containing the original transactions with all public items replaced by $\tilde{i}$, each iteration involves replacing a generalized item $\tilde{i}$ with two less generalized items $\tilde{i}_l$ and $\tilde{i}_r$. RBAT does that greedily by maximizing $UL(\tilde{i}) - (UL(\tilde{i}_l) + UL(\tilde{i}_r))$. To find less generalized items of $\tilde{i}$, each iteration performs three main steps as follows.

Step 1 (Finding a Specialization): RBAT uses a two-step heuristic to split $\tilde{i}$ into two disjoint subsets (step 4). The first step finds a pair of items from $\tilde{i}$ that incur maximum UL when generalized together. The pair is used to initialize two subsets. The second step iterates over the remaining items of $\tilde{i}$, generalizing each item with the items of either subset based on which generalization incurs less information loss.

Step 2 (Data Specialization): After the split, the dataset is updated to take the effect of specialization (step 5). Given that $\tilde{i}$ is split into $\tilde{i}_l$ and $\tilde{i}_r$, the algorithm creates a temporary dataset D' which is a copy of the current anonymized data $\tilde{D}$ but with all occurrences of $\tilde{i}$ replaced by either $\tilde{i}_l$ or $\tilde{i}_r$.

Step 3 (Checking for Privacy Protection): Finally, the check phase (step 6) ensures that current specialization does not compromise the required privacy level. It checks the protection of all PS-rules and returns true if all PS-rules are protected in D', in which case D' is copied to $\tilde{D}$ and the items $\tilde{i}_l$, $\tilde{i}_r$ are enqueued to be considered for further split (steps 7-9). If check returns false, then the current split is discarded and $\tilde{i}$ is not considered for further split.

The algorithm continues the above three main steps until Q contains no generalized items to be considered for further split and returns $\tilde{D}$ in the end (step 12). This top-down specialization process constructs a binary tree called Split-Tree with root representing the most generalized item and the set of leaf nodes called Split-Cut. The generalized items represented by a Split-cut can not be specialized further without violating any privacy constraints.

Example 3. *Consider the anonymization of the transaction dataset shown in Table 3.1 with $k = 3, c = 0.6$ and a set of PS-rules $\Theta = \{be \rightarrow gh, f \rightarrow l\}$. Initially, all the* *public items in D are generalized to a single most generalized item $\tilde{i} = (a, b, c, d, e, f)$. The generalized dataset $\tilde{D}$ produced at this stage is shown in Table 3.2.*

The split phase attempts greedily to find two more specific representations of $\tilde{i} = (a, b, c, d, e, f)$ in order to minimize UL. Since $\langle a, e\rangle$ is the pair in $\tilde{i}$ incurring maximum information loss when generalizing together, the algorithm uses it as seeds to initialize two subsets. The remaining items $\{b, c, d, f\}$ are iterated by the method in the same sequence as they appear in $\tilde{i}$, and each item is generalized to either of the subsets depending on which generalization incurs less information loss. This results in two specialized items i.e. $\tilde{i}_l = (a, b, f)$ and $\tilde{i}_r = (c, d, e)$ being returned.

To ensure that the current split gives the required privacy level, a temporary dataset D' is created. D' contains a copy of $\tilde{D}$ with $\tilde{i}$ replaced by $\tilde{i}_l$ and $\tilde{i}_r$. That is, a, b and f are replaced by $\tilde{i}_l = (a, b, f)$ where as c, d and e are replaced by $\tilde{i}_r = (c, d, e)$. D' is then used in the check phase. The check phase returns true, since $\phi(b)\phi(e) = (a, b, f)(c, d, e)$ and $\phi(f)$ are supported by more than $k = 3$ transactions in D' and the confidence $Conf(be \rightarrow gh) = 0.5$ and $Conf(f \rightarrow l) = 0.5$ is also not greater than $c = 0.6$.

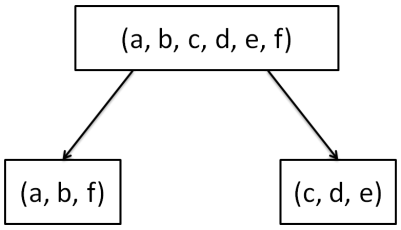


Figure 3.1: An example Split-Tree

After the check phase returns true, the generalized dataset $\tilde{D}$ and $\tilde{P}$ are updated. That is, $\tilde{D}$ is updated to contain the dataset as shown in Table 3.3 and $\tilde{P}$ is updated by replacing $\tilde{i} = (a, b, c, d, e, f)$ with $\tilde{i}_l = (a, b, f)$ and $\tilde{i}_r = (c, d, e)$. The split-update-check iteration is repeated for the (a, b, f) and (c, d, e). Let splitting $\tilde{i}_l = (a, b, f)$ and $\tilde{i}_r = (c, d, e)$ result in $\{(a, f), (b)\}$ and $\{(c, d), (e)\}$ respectively. Note that items b and e are generalized to themselves. Now, the PS-rule $be \rightarrow gh$ is supported by 2 *transaction records, and hence will not remain protected after specializing $\tilde{i_l}$ and $\tilde{i_r}$ further. Therefore, the algorithm return $\tilde{D}$ (Table 3.3). The Split-Tree of the anonymization process is shown in Figure 3.1 with root representing the most generalized item i.e.* (a, b, c, d, e, f) *and the Split-Cut containing the current set of mappings i.e.* $\{(a, b, f), (c, d, e)\}$.

Diagnosis Codes
(a,b,f), (c,d,e), **g**, **h**
(a,b,f), (c,d,e), **i**, **j**
(a,b,f), **l**
(a,b,f), (c,d,e), **g**, **h**
(c,d,e), (a,b,f), **l**

Table 3.3: An anonymized dataset $\tilde{D}$

3.3 Parallel Design of RBAT using MapReduce

There can be different ways to parallelize RBAT. Our goal is to achieve scalability for large problem instances, but this must not be at the cost of high utility loss and the loss of privacy level provided by RBAT. Therefore, we consider the solutions that preserve the three key (iterative) steps of the heuristics used by RBAT. Figure 3.2 gives a general framework for parallelization of the split, update and check phases of RBAT using MapReduce.

As shown by Figure 3.2, the parallel solution is iterative. Each iteration consists of a number of MapReduce rounds and performs the operations of split, update and check in parallel. We consider parallelizing these steps by partitioning the inputs to RBAT and performing computations on them in parallel.

There are different inputs to RBAT which may be considered for parallelization. Since

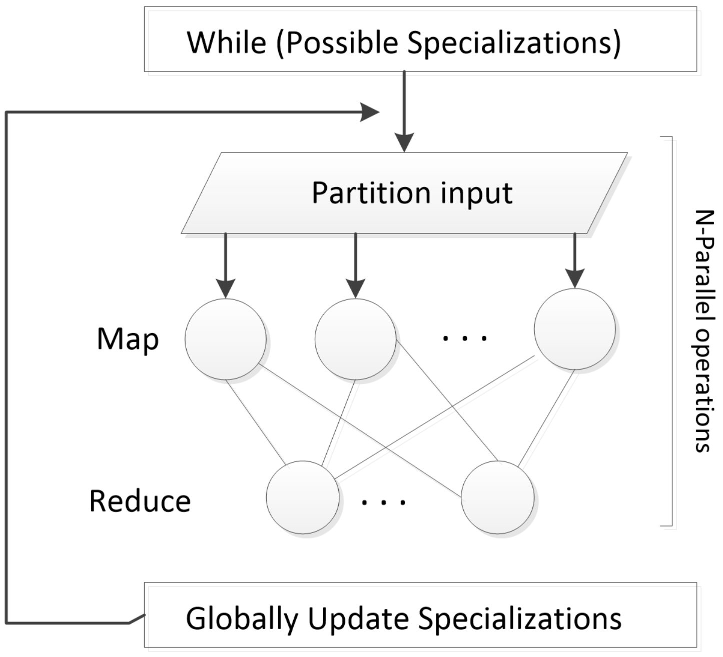

While (Possible Specializations)
Partition input
Map
Reduce
N-Parallel operations
Globally Update Specializations

Figure 3.2: A general framework for MapReduce-based design of RBAT

MapReduce assumes that the input considered for partitioning comes from a single source, partitioning more than inputs is not supported by the model [23]. One way is to partition the transactions only. This will allow the algorithm to deal with large data anonymization but partitioning transactions only will not be sufficient. Some operations of the algorithm may not require to perform computations on D and therefore partitioning D will not introduce any parallelism in such operations. For example, PS-rules are checked using a temporary anonymized version D' of D. Another reason is that the method may not be able to deal with a large domain of items and a large number of privacy constraints.

Another input to consider for partitioning is PS-rules. When anonymizing a transaction dataset, each specialization may require a large number of PS-rules to be checked at each iteration. So partitioning PS-rules may also be considered in our parallel design. This will allow the algorithm to deal with large privacy constraints. Assuming that each machine will have access to the whole dataset, the rule checking can be performed locally on each machine and there will be no shuffle cost. However, this will limit parallelization to the check phase only because the split phase does not have PS-rules as input. Such partitioning may not also allow to deal with large datasets because each machine will require access to the whole dataset.

Transaction datasets can consist of a large domain of items. For example, some of the real-world datasets such as the dataset released by AOL contained $10,154,742$ unique search queries and the records of $657,426$ users [97]. So partitioning D may not reduce the overall memory requirement. For example, consider the first split when $\tilde{i} = P$ and assume that D is partitioned. Finding a pair with maximum UL will require memory at O(P^2). Therefore, partitioning the domain of items may also need to be considered.

Preserving the key steps of RBAT in its parallel design is also challenging due to limitations of MapReduce (discussed in the previous chapter). In RBAT, these key steps are performed in sequence repetitively. That is, each split-update-check iteration requires more than one MapReduce round. Therefore, the parallelism overhead incurred by each iteration may be high. Also, most of the operations of RBAT require a global view of data. If the partitioning is not made carefully, high communication cost may be incurred. For example, if each mapper requires to shuffle most of the transaction records to the reduce stage, the communication cost of the MapReduce round will be high especially when the size of the dataset is very large.

In Chapters 4 and 5, we further discuss these issues in detail and will attempt to address them in our parallel design.

3.4 Summary

In this chapter, we discussed the concepts important to understanding our solution in the following chapters. We presented a general framework to understand the design our parallel method, and discussed different ways to perform partitioning and MapReduce operations. Our discussion showed that uniformly partitioning one input only through all MapReduce-based anonymization operations (*e.g.* partitioning only Θ, domain of items or D) may not lead to a scalable solution for all problem instances. Preserving the key steps of RBAT and their sequence in its MapReduce-based parallelization may also cause a large amount of parallelism overhead. Therefore, a more careful way to parallelize RBAT is required. In the next two chapters, we analyse these issues in detail and will discuss the solutions to address them in our parallel design.

Chapter 4

Direct Parallelization of RBAT

In this chapter, we describe our first attempt to parallelize RBAT into a sequence of MapReduce operations. We focus on the partitioning of input in order to make RBAT scalable to large problem instances.

As discussed in the previous chapter, there are different partitioning strategies that can be considered. We partition data D and the associated domain of items here, while assuming Θ to be small enough to fit into the memory of single machine. This is because while partitioning PS-rules can help improve performance, its size is unlike to be huge in practice, and our focus in this direct parallelization of RBAT is to make it scalable to large datasets.

Observing that the support computation required by the split and check phases (Steps 4 and 6 of Algorithm 3.1) and the update phase (Step 5) of RBAT, needs the whole dataset to be scanned. Our solution partitions the data across the available computing nodes and performs these operations in parallel. The data partitions created are all disjoint. That is, no transaction records are replicated to more than one partition. Our partitioning allows to control the number of distinct items in each partition. To create the parallel tasks to be performed independently, our method only requires a small amount of input to be replicated to each mapper. For example, PS-rules are copied to all mappers at the check phase. In the following, we explain our method in detail. We describe our data partitioning approach and parallel support computation in Section 4.1 and 4.2 respectively, followed by the parallel MapReduce based design of RBAT in Section 4.3. The notations used throughout our work are shown in Table 4.1.

Notation	***Description***
D	Original Dataset
$\tilde{D}$	Anonymized Dataset
Θ	PS-rules
I	Domain of items
P	Public items
S	Sensitive items
$\tilde{P}$	Split-Cut
$\tilde{i}$	Generalized item under specialization
k	Minimum support for each PS-rule
c	Maximum confidence for each PS-rule
M	Maximum number of available mappers
R	Maximum number of available reducers

Table 4.1: Notations used in our work

4.1 Data Partitioning and Representation

There are two important aspects that must be considered when designing a MapReduce-based parallel solution. First, the partitioning must be made in a way that the workload is balanced and replication is minimized among slave nodes. Second, the data must be represented in a way that computations can be performed efficiently. Given a set of transactions D to be anonymized and M mappers available, we partition D among M mappers in such a way that each mapper gets approximately an equal number of records. Generally speaking, there are two approaches to partitioning and representing

D: item-based or record-based.

With item-based partitioning, a set of transactions is partitioned vertically based on items. Let I be the domain of all items in D. We divide I into M disjoint subsets $I_j, 1 \leq j \leq M$, and then hash D into M subsets based on I_j. Each mapper m is assigned a set of pairs $\langle i, v \rangle$, where i is an item in I_j and v is a vector of identifiers of transactions that contain i as an item. The assigned pairs are represented as the following hash table:

$$\begin{pmatrix} i_1 & : & t_{1,1} & \cdots & t_{1,\beta_1} \\ i_2 & : & t_{2,1} & \cdots & t_{2,\beta_2} \\ \dots & & \dots & \dots & \dots \\ i_{|I|_j} & : & t_{|I|,1} & \cdots & t_{|I|,\beta_{|I|}} \end{pmatrix}$$

where $i_1, \dots, i_{|I|} \in I$ serve as index, and $t_{j,l}, 1 \leq j \leq |I|, 1 \leq l \leq \beta_j$ are identifiers of transactions containing the index item. β_j ($1 \leq \beta_j \leq |D|$) represents the size of j_{th} vector. For example, using item-based representation, Table 4.2 can be represented as follows.

TID	*Purchased items*
1	b, c, **f**,
2	a, d, **f**
3	a, f
4	b, d, e, **f**
5	d, f

Table 4.2: An example dataset T

$$\begin{pmatrix} a & : & 2 & 3 & & \\ b & : & 1 & 4 & & \\ c & : & 1 & & & \\ d & : & 2 & 4 & 5 & \\ e & : & 4 & & & \\ f & : & 1 & 2 & 4 & 5 \end{pmatrix}$$

Item-based partitioning is efficient for support computation. For example, suppose that the item-based representation (shown above) of the dataset shown in Table 4.2 is divided into two partitions I_1 and I_2. Let I_1 be assigned to a mapper and contain the vectors corresponding to the items a, b and c. To compute $\sigma_T(a, b)$, the method will only need to access the vectors indexed by a and b. But when the vectors corresponding to all the items of an itemset are assigned to different partitions, the support computation can not be performed locally at any mappers and may need to shuffle the relevant vectors over the network to a reducer. For example, consider the same partitioning used in the previous example. Computing the support of (a, d), will require both mappers to shuffle the relevant vectors over the network. This may dramatically increase the shuffle cost, specifically when the data is large.

Record-based partitioning, on the other hand, partitions data horizontally by records. Given D has $\{t_1, \ldots, t_{|D|}\}$ transactions, it assigns about $n = \frac{|D|}{M}$ transaction records to each mapper. That is, $\{t_1, \cdots, t_n\}$ are assigned to the first mapper, $\{t_{(n+1)}, \cdots, t_{2n}\}$ to the second mapper, and so on. Note that for efficiency purposes, we assign transactions based on their arrival order, as it outperformed other two approaches discussed in [31]. However, our method can trivially be adapted to any other approach discussed in [31] such as random partitioning. That is, to randomly assign the transaction records to partitions.

Using the record-based partitioning approach, each mapper will have a disjoint subset of transaction records $D_j, 1 \leq j \leq M$, where D_j may contain any items of I. The assigned set of records is represented as follows:

$$\begin{pmatrix} t_1 & : & i_{1,1} & \cdots & i_{1,\alpha_1} \\ t_2 & : & i_{2,1} & \cdots & i_{2,\alpha_2} \\ \cdots & & \cdots & \cdots & \cdots \\ t_{|D_j|} & : & i_{|D_j|,1} & \cdots & i_{|D_j|,\alpha_{|D_j|}} \end{pmatrix}$$

where $i_{j,l}$-th ($1 \leq j \leq |D_j|$, $1 \leq l \leq \alpha_j$) entry represents the l_{th} item of j_{th} transaction record, and α_j ($i \leq \alpha_j \leq |D|$) represents the number of items in j_{th} transaction record. This approach will only require count to be distributed over the network. For example, consider the support computation of (ab) using the record-based representation of T shown in Table 4.2. Let T be partitioned into two subsets with T_1 containing the first three transaction records and T_2 containing the last two transaction records. The support computation from T will only require $\sigma_{T_1}(ab)$ and $\sigma_{T_2}(ab)$ be computed and sent to a single reducer. Therefore, the network cost does not grow with the size of data to be anonymized. However, the support computation of a generalized item, using such representation will require the scanning of the whole partition.

We may leverage a hybrid of both techniques for data partitioning and representation. Adopting record-based partitioning can potentially reduce the network cost which could potentially be far more significant than performing an in-memory scan of the data partition. So we assign each partition D_j to a mapper, and each mapper (in parallel to other mappers) reads its assigned subset for processing. To make the support computation efficient, we can represent each data partition D_j using the hashing structure employed by item-based partitioning. This approach will ensure low overhead for shuffling the data over the network and will also make the support computation efficient. But observe that the memory required by each partition may still be large due to the number of items in each partition. To demonstrate this, consider D with the domain size $|I|$ is partitioned among M mappers. Assume that each partition is assigned $\frac{|D|}{M}$ transaction records and each item of I appears at least once in each data partition. The memory requirement on each mapper will be O($\frac{|D|}{M} \times |I|$). When $|I|$ is large, each partition may still require a large amount of memory.

One way to handle a large domain of items is to construct a partitioning method that allows to control the number of distinct items in each partition. To ensure this, we represent the dataset using a matrix with $|D|$ rows and $|I|$ columns. The rows are labelled with the transaction identifers and the columns are labelled with the items. An element $a_{i,j} \in \{0, 1\}$ is 1 if j_{th} item exists in i_{th} transaction and 0 otherwise. For example, the matrix representation of the transaction dataset shown in Table 4.2 is as follows.

$$\begin{array}{cc} & \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix} \end{array}$$

Using the matrix representation, we partition data using a two-way partitioning approach: horizontal and vertical so that each partition contains a part of the whole domain. For example, the above matrix can be divided into four partitions as follows.

$$\begin{array}{cc} & \begin{array}{ccc|ccc} a & b & c & d & e & f \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \left(\begin{array}{ccc|ccc} 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{array}\right) \end{array}$$

Such partitioning allows to control the number of transaction records and the number of distinct items in each partition. Once the data is partitioned using the two-way partitioning approach, we represent each partition using an item-based representation. For example, the first partition containing the first three transaction records and the columns corresponding to the items $\{a, b, c\}$ can be represented as follows:

$$\begin{array}{c} \\ a \\ b \\ c \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} \end{array}$$

For simplicity of our discussion, we now assume that given $|D| \times |I|$ matrix representation of D is divided into $\{D_1, \ldots, D_{\sqrt{M}}\}$ horizontally and into $\{I_1, \ldots, I_{\sqrt{M}}\}$ vertically and we use the notation $\{D_1, \ldots, D_M\}$ to represent such partitioning. In general, the number of partitions made vertically do not necessarily have to be the same as the horizontal number of partitions.

4.2 Parallel Support Computation

We observe that support computation is the most frequent operation in RBAT that requires full data scan. It is performed to compute the UL of generalized items and to check if PS-rules are protected. In RBAT, the support is computed using Definition 1.

We now show how support is computed using MapReduce. Given D and a set $\lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_h\}$ where each λ_j $(1 \leq j \leq h)$ is an itemset, the support of all itemsets in λ can be computed using a single MapReduce round. D is partitioned into $\{D_1, \ldots, D_M\}$ using the partitioning approach discussed in the previous section. Each mapper then iterates over λ, computing and emitting the local support of each element $\lambda_a \in \lambda$:

$$Map(D_j, \lambda_1, \ldots, \lambda_{|\lambda|}) \quad \rightarrow \quad [\langle \lambda_1, \sigma_{D_j}(\lambda_1)\rangle \ldots \langle \lambda_{|\lambda|}, \sigma_{D_j}(\lambda_{|\lambda|})\rangle]$$

These partial supports are shuffled over the network to the corresponding reducers. Each reducer (in parallel to other reducers) accumulates the partial supports corresponding to λ_x ($1 \leq x \leq |\lambda|$) and computes the global support.

$$\mathit{Reduce}([\langle \lambda_x, \sigma_{D_1}(\lambda_x) \ldots, \sigma_{D_j}(\lambda_x) \rangle] \quad \rightarrow \quad \langle \lambda_x, \sum_{j=1}^{M} \sigma_{D_j}(\lambda_x) \rangle$$

4.3 Direct Parallelization of RBAT

In this section, we will discuss our parallel design of RBAT using MapReduce. We mainly parallelize RBAT by partitioning data and performing all the required support computations in parallel using the partitioning strategy and support computation pattern discussed in Sections 4.1 and 4.2 respectively. A schematic representation of our parallel RBAT is shown in Figure 4.1. Our method assumes that the privacy constraints are small enough to fit into the memory of a single machine.

Preprocessing: The algorithm starts with a pre-processing phase, which computes the pairwise UL of all possible pairs of P. The pseudocode of this phase is shown in Algorithm 4.1. Steps (2-10) compute the support of all the pairs, using a single MapReduce round. In addition to computing the global support of each pair, the reduce stage also computes the pairwise UL of all the pairs and outputs $\mathcal{P}$ where $\mathcal{P}$ is a $|P| \times |P|$ matrix with the rows and columns labelled by the items in P and an entry $\mathcal{P}_{i,j}$ ($1 \leq i, j \leq |P|$) gives $UL(i, j)$.

Note that the UL measure (Definition 5) is symmetric *e.g.* $UL(i_x, i_y) = UL(i_y, i_x)$, so we only calculate the UL of the pairs corresponding to the entries of upper or lower triangle only. Let $|P|$ denote the size of P. The number of distinct pairs will be $\binom{|P|}{2} = \frac{1}{2} \cdot (|P| - 1) \cdot |P|$. Assuming that each mapper is assigned the same number of transaction records and the number of items in each partition is also the same, the memory requirement for each mapper to hold its assigned data partition is O($\frac{|D|}{M}$).

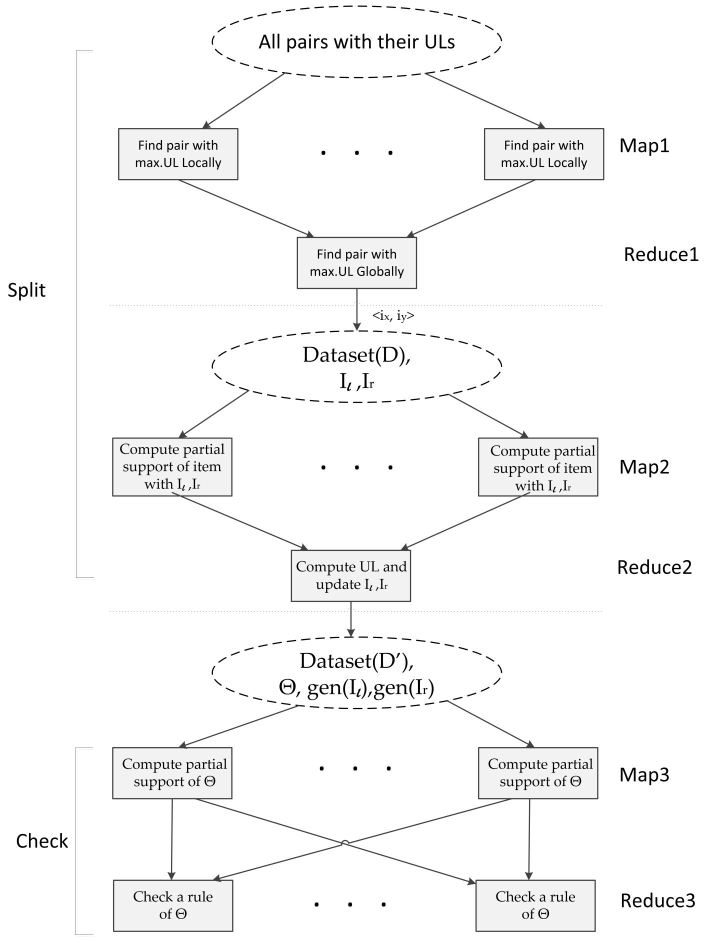

All pairs with their ULs
Find pair with max.UL Locally
Find pair with max.UL Locally
Map1
Find pair with max.UL Globally
Reduce1
Split
<ix, iy>
Dataset(D), Iₗ ,Iᵣ
Compute partial support of item with Iₗ ,Iᵣ
Compute partial support of item with Iₗ ,Iᵣ
Map2
Compute UL and update Iₗ ,Iᵣ
Reduce2
Dataset(D'), Θ, gen(Iₗ),gen(Iᵣ)
Compute partial support of Θ
Compute partial support of Θ
Map3
Check
Check a rule of Θ
Check a rule of Θ
Reduce3

Figure 4.1: A schematic representation of Parallel RBAT

Each mapper also creates and holds $(\frac{(|P|-1)|P|}{2})$ output pairs. Therefore, the amount of memory required by a mapper on each machine will be $O(\frac{|D|}{M} + \frac{(|P|-1)|P|}{2})$. For a mapper, the cost of creating the index-based representation of its assigned data partition is $O(\frac{|D|}{M})$. Considering that each mapper computes the partial support of all pairs on data partitions in parallel, the overall processing cost of the map stage consists of the cost involved in representing the partitions using item-based scheme and computing the partial support of all pairs, and is $O(\frac{|D|}{M} + \frac{(|P|-1)|P|}{2})$.

Here, the number of distinct keys in the output pairs produced by the map stage is $\binom{|P|}{2}$. If the key space is partitioned into R equal partitions, and all R reducers copy and process the map output in parallel, the shuffling cost in the worst case (when no reducer resides locally to its assigned map output) and computational cost of the reduce stage is O($M \cdot \frac{(|P|-1)|P|}{2R}$).

Algorithm 4.1: COMPUTEUL (P)

Input: A data partition D_j, Domain of public items P.

Output: A matrix $\mathcal{P}$, containing all pairwise ULs.

```
1:  Map($m, \langle D_m, P \rangle$)
2:    $D_m \leftarrow$ Load the $m$-th partition of $D$ from DFS
3:    $i_x \leftarrow \varnothing$, $i_y \leftarrow \varnothing$
4:    for $x \leftarrow 1$ to $(|P| - 1)$ do
5:      $i_x \leftarrow x$-th item of $P$.
6:      for $y \leftarrow (x+1)$ to $|P|$ do
7:        $i_y \leftarrow y$-th item of $P$.
8:        EMIT $\langle (i_x\, i_y), \sigma_{D_m}(i_x, i_y) \rangle$
9:      end for
10:   end for
11: Reduce($(i_x\, i_y), [\langle \sigma_{D_1}(i_x, i_y) \rangle, \langle \sigma_{D_2}(i_x, i_y) \rangle, \ldots, \langle \sigma_{D_M}(i_x, i_y) \rangle]$)
12:   $\mathcal{P}[x][y] \leftarrow UL(i_x, i_y)$
13:   EMIT $\langle (x\, y), \mathcal{P}[x][y] \rangle$
```

Finding Maximum Pair: The first phase shown in Figure 4.1 corresponds to the first step of the split phase (Step 4) of RBAT. Given a generalized item $\tilde{i}$, the algorithm uses a single MapReduce round with M mappers and a single reducer to find a pair which when generalized together incurs maximum UL. Each mapper reads a subset of a pre-computed matrix $\mathcal{P}$ containing ULs of all possible pairs of public items P (Step 2). While reading its assigned subset $\mathcal{P}_m$, each mapper keeps track of the pair with maximum UL found so far, and shuffles to a single reducer in the end (Step 3). Once all the locally found pairs with the maximum UL arrives, the reducer compares them to find the pair $\langle i_x, i_y \rangle$ with maximum UL globally. Since a mapper keeps the pair with maximum UL only, the memory requirement is constant irrespective of the size of $\mathcal{P}$. Considering that M available mappers read $\mathcal{P}$ in parallel, the overall computational cost of the map phase is O$(\frac{|\mathcal{P}|}{M})$. Also, each mapper outputs only one pair, the maximum number of pairs shuffled to a reducer is M. Therefore, the amount of memory required at the reduce phase is O(M).

Note that the algorithm reads all pairs and their UL in $\mathcal{P}$ but certain pairs can be ignored without comparison *i.e.* a pair containing an item which is not generalized to $\tilde{i}$. For example, consider $P = \{a, b, c, d\}$ and the matrix $\mathcal{P}$ shown as follows:

$$\mathcal{P} = \begin{pmatrix} a & UL(a,b) & UL(a,c) & UL(a,d) \\ b & & UL(b,c) & UL(b,d) \\ c & & & UL(c,d) \end{pmatrix}$$

At the first split, the most generalized item is to be specialized so all four items of P are generalized to $\tilde{i} = (a, b, c, d)$. Splitting $\tilde{i}$ requires comparison of all pairs in $\mathcal{P}$. Suppose $\tilde{i}$ is specialized to $\tilde{i}_l = (a, b, c)$ and $\tilde{i}_r = (d)$. When splitting $\tilde{i}_l$, the mappers will only need the pairs containing a, b and c. So the pairs such as (a, d) or (c, d) are not considered for comparison and ignored.

Algorithm 4.2: FINDMAXPAIR ($\tilde{i}$)

Input: A generalized item $\tilde{i}$ to split.

Output: A pair $i_x, i_y \in \tilde{i}$ such that $UL(i_x, i_y)$ is maximum.

```
1: **Map**$(m, \mathcal{P}_m)$
2:     $\mathcal{P}_m \leftarrow$ Load the $m$-th partition of $\tilde{P}$ from DFS
3:     EMIT($\varnothing$, $\langle i_a, i_b \rangle$) such that $UL(i_a, i_b)$ is maximum in $\mathcal{P}_m$
4: **Reduce**($\varnothing$, $[\langle i_{a_1}, i_{b_1} \rangle, \langle i_{a_2}, i_{b_2} \rangle, \ldots, \langle i_{a_M}, i_{b_M} \rangle]$)
5:     $\langle i_x, i_y \rangle \leftarrow \langle i_{a_j}, i_{b_j} \rangle$ such that $UL(i_{a_j}, i_{b_j})$ is maximum, $1 \le j \le M$
6:     EMIT($\varnothing$, $\langle i_x, i_y \rangle$)
```

Split using seeds: The second step of the split phase is iterative (Steps 2-16 of Algorithm 4.3) and uses $\langle i_x, i_y \rangle$ to split $\tilde{i}$ into two less generalized items $\tilde{i_l}$ and $\tilde{i_r}$. Initially, I_l and I_r are assigned the seeds i_x and i_y (Step 2). For every item $i_z \in \tilde{i}$ that is not a seed, a MapReduce round is used to decide whether it should be generalized with I_l or I_r (Steps 3-15). The M mappers read D in parallel and compute $\sigma_{D_j}(I_l \cup \{i_z\})$ and $\sigma_{D_j}(I_r \cup \{i_z\})$. These partial supports from all the mappers are shuffled over the network to a single reducer (Steps 4-7). The reducer then computes the UL of $I_l \cup \{i_z\}$ and $I_r \cup \{i_q\}$ and adds i_z to either I_l or I_r based on which generalization incurs less UL (Steps 8-14). Finally, I_l and I_r are returned as two less generalized items (Step 16).

For example, consider the split of $\tilde{i} = (a, b, c, d)$. Suppose (a, d) is the pair incurring maximum UL on generalization. Let $I_l = \{a\}$ and $I_l = \{d\}$, the first MapReduce round generalizes b either with the items of I_l or I_r. Suppose $D = \{D_1, D_2\}$, the mappers compute $\sigma_{D_1}(a, b)$, $\sigma_{D_2}(a, b)$, $\sigma_{D_1}(b, d)$ and $\sigma_{D_2}(b, d)$. The reducers aggregates the partial supports and compute $UL(a, b)$ and $UL(b, d)$. Suppose $UL(a, b) > UL(b, d)$, I_r is updated to contain b and d. In the next MapReduce round, the algorithm computes the $UL(a, d)$ and $UL(b, c, d)$ and assuming that $UL(b, c, d) > UL(a, d)$, the algorithm generalizes d to I_l and return the generalized items $\tilde{i_l} = (a, d)$ and $\tilde{i_r} = (b, c)$.

Using such item-based representation, the partial support computation of an item from a data partition is O(1). In order to compute the partial support $\sigma_{D_j}(I_l \cup \{i_z\})$ and $\sigma_{D_j}(I_r \cup \{i_z\})$, the number of indexed vectors to be accessed is $(|\tilde{i_l}| + 1)$ and $(|\tilde{i_r}| + 1)$. Given that $I_l \cap I_r = \varnothing$, the computational cost of the map stage is O$(\tilde{i})$.

At each round of this stage, a mapper needs to hold a data partition of size $(\frac{|D|}{M})$ and the generalized items $\tilde{i_l}$ and $\tilde{i_r}$. So the overall memory required for a map task on each machine is O$(\frac{|D|}{M} + |P|)$. Note that each mapper outputs only one pair, containing the partial supports $\sigma_{D_j}(I_l \cup \{i_z\})$ and $\sigma_{D_j}(I_r \cup \{i_z\})$. Therefore, the memory required for the output at each MapReduce round is O(1). Since each mapper communicates one pair only, the shuffle cost is O(M) at each MaReduce round. At this stage, all pairs are received and processed by one reducer only. Therefore, the computational cost of reduce stage is also O(M).

Algorithm 4.3: MR-SPLIT $(\tilde{i})$

Input: A generalized item $\tilde{i}$ to split.

Output: Less generalized items $\tilde{i_l}$ and $\tilde{i_r}$

```
1:  $\{i_x, i_y\} \leftarrow$ FINDMAXPAIR $(\tilde{i})$
2:  $I_l \leftarrow i_x, I_r \leftarrow i_y$
3:  **for** each $i_z \in \tilde{i}$ and $z \notin \{x, y\}$ **do**
4:    **Map**($m$, $\langle I_l$, $I_r$, $i_z$, $D_m\rangle$)
5:      $D_m \leftarrow$ Load the $m$-th partition of $D$ from DFS
6:      $\tilde{i_l} \leftarrow I_l \cup \{i_z\}, \tilde{i_r} \leftarrow I_r \cup \{i_z\}$
7:      EMIT($i_z, \langle\sigma_{D_m}(\tilde{i_l}), \sigma_{D_m}(\tilde{i_r})\rangle$)
8:    **Reduce**($i_z, [\langle\sigma_{D_1}(\tilde{i_l}), \sigma_{D_1}(\tilde{i_r})\rangle, \langle\sigma_{D_2}(\tilde{i_l}), \sigma_{D_2}(\tilde{i_r})\rangle, \ldots, \langle\sigma_{D_M}(\tilde{i_l}), \sigma_{D_M}(\tilde{i_r})\rangle]$)
9:      **if** $UL(\tilde{i_l}) > UL(\tilde{i_r})$ **then**
10:       $I_l \leftarrow I_l \cup \{i_z\}$
11:     **else**
12:       $I_r \leftarrow I_r \cup \{i_z\}$
13:     **end if**
14:     EMIT($\langle I_l, I_r\rangle$)
15: **end for**
16: **return** $\langle\tilde{i_l} = I_l, \tilde{i_r} = I_r\rangle$
```

Update: As discussed in Section 3.2, RBAT undergoes the update phase after each split and uses two less generalized items to create a temporary dataset D'. Let us assume that $\tilde{i}$ is split into $\tilde{i_l}$ and $\tilde{i_r}$. Suppose the anonymized form $\tilde{D}$ of D is partitioned among M mappers. Given a data partition $\tilde{D}_j$, a mapper replaces all occurrences of $\tilde{i}$ with $\tilde{i_l}$ and $\tilde{i_r}$. Finding out if an occurrence of $\tilde{i}$ in a transaction record of $\tilde{D}$ is to be replaced by $\tilde{i_l}$ or $\tilde{i_r}$, requires access to the original representation D_j of $\tilde{D}_j$. For example, consider a transaction record $t = \{a, b, c, \mathbf{h}\}$, its anonymized form $\tilde{t} = \{a, (b, d, f), c, \mathbf{h}\}$ and that the generalized item $\tilde{i} = (b, d, f)$ is split into $\tilde{i_l} = (b, d)$ and

$\tilde{i}_r = f$. Creating an updated version of $\tilde{t}$ requires replacing $\tilde{i}$ with $\tilde{i}_l$ or $\tilde{i}_r$. Since, the original representation of the item in t is b, which is now mapped to $\tilde{i}_l$ so the updated version is $t' = \{a, (b, d), c, \mathbf{h}\}$.

This requires to hold three versions of a dataset. Considering that D' is discarded if the PS-rules are unprotected, one possible way to reduce the space requirement is to update the generalized dataset $\tilde{D}$ after checking the privacy constraints. The algorithm will not need to construct a temporary dataset and no specializations will be needed to roll back on account of inadequate privacy. But, the check phase will require the generalized dataset $\tilde{D}$ as well as its original counterpart D. Our approach is to keep the current split-cut, instead of constructing and updating $\tilde{D}$.

We divide the update phase into two parts. The first part (depicted in Algorithm 4.4) is performed after each split, and uses a single map-only round to update the current split-cut. The current split-cut $\tilde{P}$ is partitioned into M subsets and each subset $\tilde{P}_m$ is loaded into the memory of mapper m (Step 2). The mapper (in parallel to other mappers) iterates over all the generalized items in $\tilde{P}_m$, to find $\tilde{i}$. If $\tilde{i}$ is found, the mapper replaces it with the specialized items $\tilde{i}_l$ and $\tilde{i}_r$ and returns the updated split-cut partition $\tilde{P}'_m$ (Steps 3-8). If the mapper does not find $\tilde{i}$ in its assigned partition then $\tilde{P}_m$ is returned (Step 9).

The second part (Algorithm 4.5) is performed once at the end of the anonymization, when no split can further be made without sacrificing the required privacy level. This is also a map-only round so incurs no shuffling cost. The mappers read the original dataset D (step 2), and are also given a copy of split-cut $\tilde{P}$. Given a data partition D_m, a mapper m generalizes each transaction record $t \in D_m$ by finding and replacing a subset $\tilde{t}$ of $\tilde{P}$ containing the mappings of all the items in t (step 4-7). The generalized form $\tilde{D}_m$ of D_m is then returned (step 8).

This will not require to store and/or load any other copy of the dataset except D. Also, the check needs the split-cut and original dataset only. Next, we also show that this will not significantly incur any extra computational or memory requirements at the check

Algorithm 4.4: UPDATECUT ($\tilde{P}$, $\tilde{i}$, $\tilde{i}_l$, $\tilde{i}_r$)

Input: A Split-cut $\tilde{P}$, generalized item $\tilde{i}$, and its specializations $\tilde{i}_l$ and $\tilde{i}_r$.

Output: The updated split-cut $\tilde{P}'$.

```
1: Map$(m, \langle \tilde{P}_m, \tilde{i}, \tilde{i}_l, \tilde{i}_l \rangle)$
2:   $\tilde{P}_m \leftarrow$ Load the $m$-th partition of $\tilde{P}$ from DFS.
3:   for each generalized item $\tilde{i}_j \in \tilde{P}$ do
4:     if $\tilde{i}_j = \tilde{i}$ then
5:       $\tilde{P}'_m \leftarrow Replace(\tilde{P}_m, \tilde{i}, \tilde{i}_l, \tilde{i}_r)$
6:       EMIT($\varnothing$, $\langle \tilde{P}_m \rangle$) and return.
7:     end if
8:   end for
9:   EMIT($\varnothing$, $\langle \tilde{P}_m \rangle$)
```

Algorithm 4.5: GENERALIZE (D, $\tilde{P}$)

Input: Original dataset D and, Split-cut $\tilde{P}$.

Output: A generalized form $\tilde{D}$ of D.

```
1: Map$(m, \langle D_m, \tilde{P} \rangle)$
2:   $D_m \leftarrow$ Load the $m$-th partition of $D$ from DFS.
3:   $\tilde{D} \leftarrow \varnothing$.
4:   for each transaction $t \in D$ do
5:     $\tilde{t} \leftarrow$ A subset of $\tilde{P}$, satisfying $\forall i \in t, \phi(i) \in \tilde{t}$
6:     Add $\tilde{t}$ to $\tilde{D}_m$
7:   end for
8:   EMIT($\varnothing$, $\langle \tilde{D}_m \rangle$)
9: return $\{\tilde{D}_1, \ldots, \tilde{D}_M\}$
```

stage.

Rule Checking: The third step shown in Figure 4.1 corresponds to the check phase (Step 3) of RBAT (see Section 3.2). To reduce the space requirements, we do not create any generalized form of D, but rather check the privacy protection using the original dataset D and the current split-cut $\tilde{P}$. Consider a PS-rule $p \rightarrow s$, an original transaction record $t \in D$ and its generalized form $\tilde{t} \in \tilde{D}$. Let $i \in P$ be a item, $\phi(i)$ be its generalized representation, and $\{\phi(i)\}$ be the set of items generalized to $\phi(i)$.

Observation 1. *If $\exists i \in p$, for which $t \cap \{\phi(i)\} = \varnothing$ holds true, then antecedent p will not be supported by $\tilde{t}$*

Observation 1 is based on the property of set-based generalization that a generalization consists of a combination of all the items mapped to it. For example, suppose that a generalized item (a, b, c) consists of a, b and c and all three items are generalized to the same generalized item. Given that a transaction record in its generalized form, supports a generalized item $\tilde{i}$ then its generalized form must contain any of the individual items of $\tilde{i}$. For example, consider a transaction record $t = \{b, d, g, \mathbf{h}\}$, its anonymized form $\tilde{t} = \{(b, c), d, (a, g), \mathbf{h}\}$ and a set of PS-rules $\Theta = \{a\ d \rightarrow h, a\ f \rightarrow h\}$. Let $\phi(f) = (e, f)$, the first PS-rule is supported by $\tilde{t}$ because $\phi(a) \cap t = g$ and $\phi(d) \cap t = d$. The second PS-rule is not supported by $\tilde{t}$ because $\phi(f) \cap t = \varnothing$.

The rule checking phase (depicted in Algorithm 4.6) employs a single MapReduce round. During the map stage, a mapper m loads its assigned data partition (step 2) into memory and also reads the current split-cut $\tilde{P}$. The mapper iterates over PS-rules Θ, computing and emitting the partial supports $\sigma_{D_m}(\tilde{p})$ and $\sigma_{D_m}(\tilde{p} \cup s)$ for each PS-rule (steps 3-5). Note that we need to compute the generalized form of each rule's antecedent and the partial support computation of each PS-rule is performed using Observation 1. The partial supports are then shuffled over the network to reducers with $p \rightarrow s$ as key. The reducers compute the global support $\sigma_{D'}(\tilde{p})$ and $\sigma_{D'}(\tilde{p} \cup s)$ for each PS-rule (step 7), and return true or false based on if the rule is protected or not (steps 8-10).

Each mapper needs to load Θ, $\mathcal{P}$, and a partition of D. The mapper also needs to hold $|\Theta|$ output pairs. Therefore, the memory required on each mapper is O($\frac{|D|}{M} + |\mathcal{P}| + |\Theta|$). Each mapper represents its assigned data partition D_j using item-based representation.

Algorithm 4.6: MR-CHECK (D, Θ, k, c)

Input: Original dataset D, PS-rules Θ, Minimum support k, Maximum confidence c.

Output: True if each protected PS-rule and False otherwise.

```
1:  **Map**($m$, $\langle D_m, \Theta\rangle$)
2:      $\mathcal{D}_m \leftarrow$ Load the $m$-th partition of $D$ from DFS
3:      **for** each rule $p \rightarrow s \in \Theta$ **do**
4:          COMPUTE AND EMIT ($p \rightarrow s, \langle \sigma_{D_m}(\tilde{p}), \sigma_{D'_m}(\tilde{p} \cup s)\rangle$) using Observation 1
5:      **end for**
6:  **Reduce**($p \rightarrow s, [\langle \sigma_{D'_j}(\tilde{p}), \sigma_{D'_j}(\tilde{p} \cup s)\rangle, 1 \leq j \leq M]$)
7:      $\sigma_{D'}(\tilde{p}) = \Sigma_{j=1}^{M} \sigma_{D'_j}(\tilde{p}), \;\; \sigma_{D'}(\tilde{p} \cup s) = \Sigma_{j=1}^{M} \sigma_{D'_j}(\tilde{p} \cup s)$
8:      **if** $\sigma_{D'}(\tilde{p}) < k$ or $\frac{\sigma_{D'}(\tilde{p} \cup s)}{\sigma_{D'}(\tilde{p})} > c$ **then**
9:          EMIT($r$, False)
10:     **end if**
```

Let the maximum size of PS-rules in Θ be $|r|$. The cost of computing partial support of all PS-rules is $(|\Theta| \cdot |r|)$. So the overall computational cost is O$(\frac{|D|}{M} + |\Theta| \cdot |r|)$. Each mapper needs to shuffle a pair, containing partial support of a PS-rule. Let R reducers copy the map output to reducers, the overall shuffle cost and the computational cost of the reduce phase is O$(M \cdot \frac{|\Theta|}{R})$.

4.4 Experimental Evaluation

In this section, we empirically analyse the scalability of our parallel design (discussed previously in Section 4.3). We measure the scalability with respect to large problem instances and the resources available. More specifically, we measured the effect of increasing cluster size, data volumes, the number of PS-rules and the domain size on the performance of our method.

We implemented our method using Hadoop version 1.0.3 [1] API combined with Java 1.7.0_8. Hadoop is the most popular and open-source implementation of MapReduce. It is also similar to Google's implementation of MapReduce [38]. A Hadoop cluster follows the master/slave architecture and consists of two main parts: data management layer, which consists of a Namenode instance running over the master node and a number of datanodes running over slave machines. The data are stored in the Hadoop Distributed File System (HDFS). Another important part is the execution engine, which consists of a Jobtracker and a set of tasktracker instances. The main execution is controlled by Jobtracker running over the masternode. All the MapReduce tasks are performed by the Tasktracker daemons running over the slave (worker) nodes.

All the experiments were performed over a cluster of fifteen computers, running Ubuntu 10.0.4 operating system and physically located within the same building. The computers were interconnected by a 100Mbps Ethernet connection. One of the machines was allocated to run the master program. All the worker machines used to perform MapReduce tasks were homogenous in configuration. The machine running the master program contains four physical cores and 8GB memory. All the slaves have 3GB physical memory, Intel Core^{TM} 2 Duo Processor Model E8500 with 6MB Cache size, and two physical cores.

We used BMS-Web-View-1 [75], a real-world click-stream dataset with $|D| = 59602$ and $|I| = 497$, in our experiments. The maximum and average transaction size of the dataset is 264 and 2.4 respectively. We acquired the larger data sizes by random selection and duplication of transaction records from the original dataset. 10% of the items were randomly selected as sensitive items. In the set of experiments, we do not report data utility, since our parallel version preserves the processing of its sequential counterpart and hence gives exactly the same anonymized results given by RBAT. Unless otherwise specified, our default settings for the experiments are given in Table 4.3

To get maximum possible performance, we followed the existing relevant work [112]

[1] http://hadoop.apache.org/

Parameter	**Default value**
$\lvert D\rvert$	32M (millions)
$\lvert I\rvert$	497
$\lvert\Theta\rvert$	1,000
k	60
c	0.8
M	32
R	8

Table 4.3: The default parameter settings for experiments

and customized the Hadoop based on the availability of resources as shown in Table 4.4.

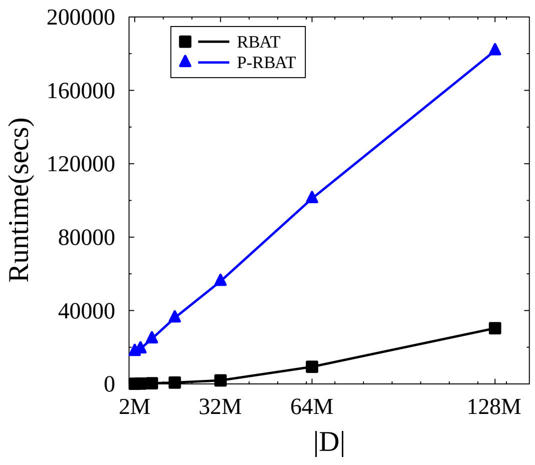


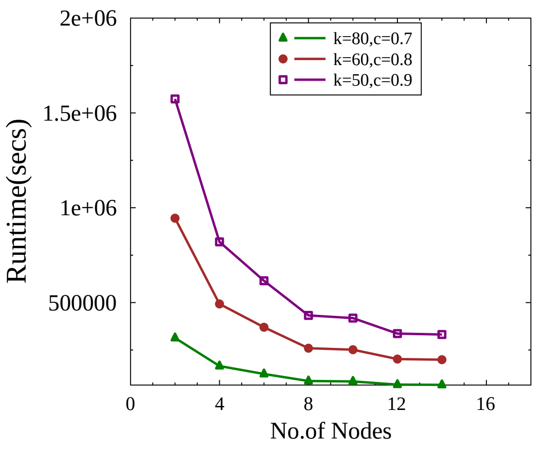
Figure 4.2: Datasize vs. Runtime

To check scalability with respect to increasing data volumes, we varied datasize from 2M to 128M (Million) transaction records and measured the response time of both RBAT and our parallel implementation. As shown in Figure 4.2, the runtime of parallel RBAT (P-RBAT) grew sub-linearly with increasing size of data and grew much slowly than RBAT did. However, P-RBAT performed less efficiently than RBAT. This is due

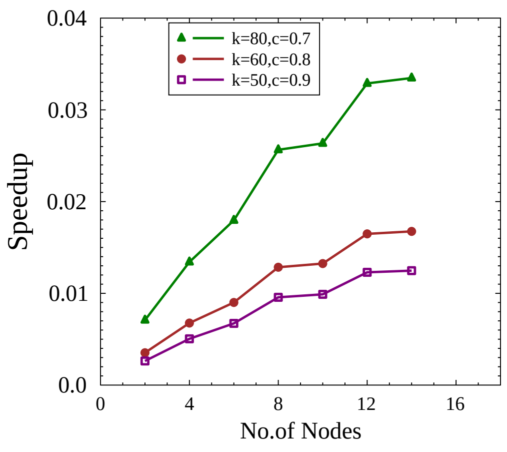
Parameter	**Description**	Value
mapred.job.reuse. jvm.num.tasks	Allows the JVM re-use among consecutive tasks	-1(yes)
io.sort.mb	Memory for sorting map output	300MB
mapred.child.ulimit	The maximum memory, of a process launched by the Map-Reduce framework	2GB
io.sort.mb	Memory for sorting map output	300MB
io.sort.spill.percent	A limit of the buffer, after which records begin to be spilled to disk	0.85
mapred.{map/reduce}. tasks.speculative.execution	Allow multiple instances of same MapReduce task to run in parallel	true
mapred.reduce.parallel.copies	The number of parallel transfers run by reduce during shuffle phase.	8

Table 4.4: Hadoop parameter settings

to the overwhelming parallelization overhead, consisting of I/O, network and setup cost incurred by each MapReduce round. We also observed that our method is less stable in terms of runtime behaviour as the dataset becomes larger. For example, increasing $|D|$ from 2M to 4M increases the runtime by a factor of 0.1, whereas from 64M to 128M, the runtime increases by a factor 0.8. This is because larger data sizes incur increased I/O cost for each data scan, hence the increased overhead which affects the scalability of our method. As described in Section 4.2, our method does not require the shuffling of any transaction records. Therefore, increasing the size of D does not affect the amount of data to be shuffled at any MapReduce round. The number of MapReduce rounds also remains fixed, provided that the domain size does not increase. So the setup and communication cost incurred by the shuffle stage of all MapReduce rounds remained fixed too. This shows that our method can handle increasing data volumes in a scalable way for dense datasets [92]. If the domain size is large, the scalability of our method can be affected by the increasing I/O cost, since the algorithm undergoes a large number of MapReduce rounds.



(a) Runtime vs Cluster Size



(b) Speedup vs Cluster Size

Figure 4.3: Scalability with respect to cluster size

We have also studied the scalability of P-RBAT with respect to the size of the computing cluster. Figures 4.3(a) and 4.3(b) show the response time and relative speed up, respectively, with varying number of computing nodes used in anonymization. The relative speedup is measured as the ratio of runtime obtained by the sequential RBAT implementation, to the runtime of P-RBAT with a cluster size whose speedup is measured. The cluster size was varied from 2 to 14. We found that P-RBAT scaled better when a small number of processing nodes were used *e.g.* 2 to 8. Setting the cluster size to more than 8 nodes did not result in any significant performance gains. This is because of the associated overhead which is proportional to the size of cluster used. For small cluster sizes, the performance gain is higher than the associated overhead and increasing the workload causes decreased workload on each machine. Therefore, our algorithm performs better until it attains maximum performance gain at certain point (at cluster size of 8 nodes in our experiment). After this point, the algorithm starts slowing down. This is because, when more nodes were used, the computation on each node is reduced, making the overhead resulting from setting up parallel processing a significant proportion of the overall response time.

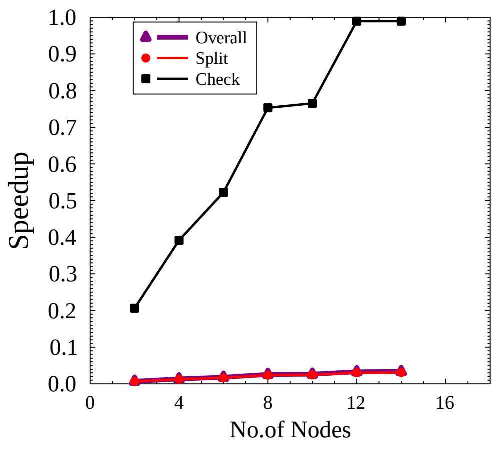
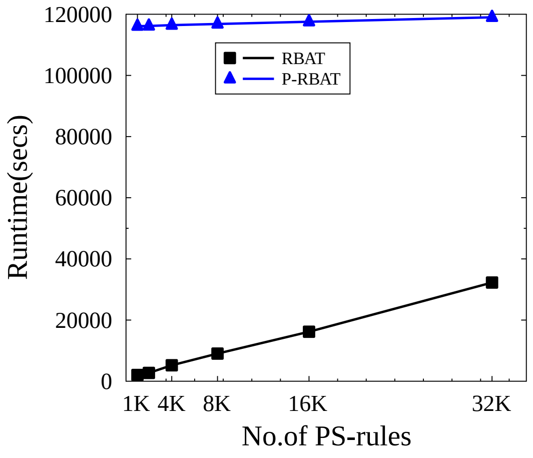


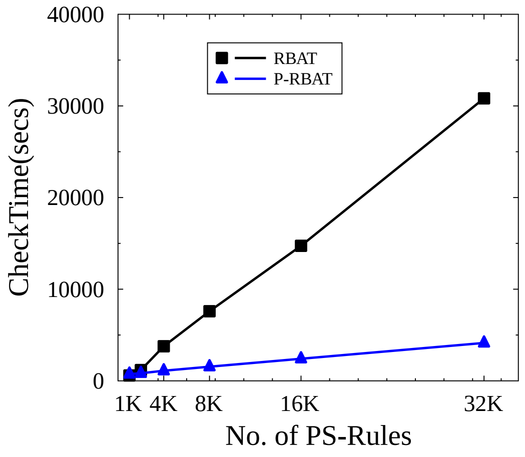
Figure 4.4: Scalability with respect to cluster size

The overall speedup is far worse than the ideal (linear) speedup. We attribute this to the parallelism overhead. Our further analysis (Figure 4.4) showed that most of the overhead came from the split phase, which is due to mapping the sequential split phase to MapReduce by only partitioning data. This affects the overall speedup of our algorithm.

We also studied the effect of privacy parameters k and c on speedup of the algorithm (Figure 4.3). Increasing c and decreasing k relax privacy requirements and allow the algorithm to undergo further splits. As shown by the results, the algorithm attains better speedup as the size of split tree grows. This indicates that most of the parallelism overhead comes from the initial splits. The number of MapReduce rounds required to split a generalized item $\tilde{i}$ increases proportionally to $|\tilde{i}|$. So the later splits are performed using fewer MapReduce rounds, and hence the overall performance gain easily offsets the parallelism overhead. This implies that the algorithm can be expected to scale better with respect to increasing cluster size, when privacy requirements are tight *i.e.* k is large and c is small.



(a) $|\Theta|$ vs. Runtime



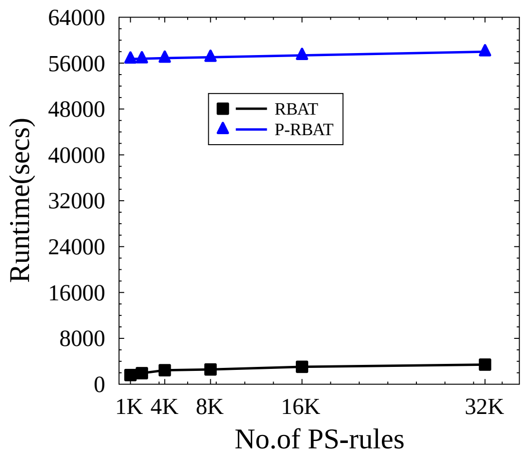
(b) $|\Theta|$ vs Check time

Figure 4.5: Performance with respect to $|\Theta|, k = 60, c = 0.8$

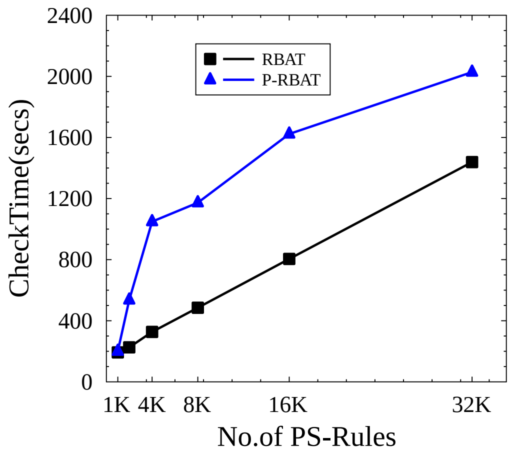
To further examine the scalability of our parallel method, we also evaluated its runtime behaviour with respect to increasing $|\Theta|$ from 1K (thousand) to 32K. As depicted in Figure 4.5, the runtime of P-RBAT grows sub-linearly to the size of Θ. P-RBAT also grows much slower in runtime than RBAT with respect to increasing $|\Theta|$, but the overall performance of P-RBAT is at least four times worse than its sequential counterpart. Figure 4.5 confirms that this is mainly because of the parallelism overhead at the split phase. Our further analyses of the effect of increasing $|\Theta|$ on the check phase (Figure 4.5-b) also show that for a smaller number of 1K to 2K PS-rules, the performance of P-RBAT's check phase is nearly same as sequential rule checking. The difference between the runtime of the two methods increases as the number of PS-rules increases. For example, when $|\Theta| = 1K$, the runtime of P-RBAT's check phase is worse than the sequential rule checking of RBAT by a factor of 30%. But, the check phase of P-RBAT at $|\Theta| = 4K$ is better by nearly 30%. So the performance of our algorithm is better, when large privacy constraints are involved.

During the check, the algorithm did not seem to utilize the resources effectively. For example, P-RBAT performs rule checking using fourteen computing nodes, but the runtime of its check phase is better than RBAT by only 7 times. To further investigate the performance bottlenecks, we examined the runtime behaviour of P-RBAT with different privacy parameter settings. Figure 4.6 shows when the algorithm undergoes a single split-check iteration. Observe here that RBAT grows more slowly and performs better than P-RBAT. Comparing the results to that in Figure 4.5, one can conclude that there is a relationship between the performance behaviour of P-RBAT and the number of split-check phases the algorithm undergoes. That is, the larger the split tree, the more the number of times the rules are checked and hence causes P-RBAT's check phase to outperform the sequential rule checking. This is also an indication that higher performance gain and better scalability can be expected when the required privacy is not high and allows the construction of a large split tree during the anonymization.

Our further analyses showed that the main bottlenecks lie in the number of PS-rules checked by P-RBAT which makes the ineffective resource utilization, because P-RBAT checked extra rules not checked by its sequential counterpart. Figure 4.7 shows the difference of the PS-rules checked by P-RBAT and the rules checked by RBAT with $k = 60$ and $c = 0.8$. Both algorithms undergo three split-check iterations. As the number of given rules Θ increases, the difference of the rules actually checked by P-RBAT and those checked by RBAT increases. This is because RBAT checks PS-rules sequen-



(a) $|\Theta|$ vs. Runtime



(b) $|\Theta|$ vs Check time

Figure 4.6: Performance with respect to $|\Theta|, k = 80, c = 0.7$

tially, and stops as soon as any rule is found unprotected. But during a MapReduce round, there is no communication among mappers until the shuffle phase. So P-RBAT can not know if any PS-rule is not protected until the end of the rule-checking round, making checking all PS-rules necessary. In the worst case, P-RBAT checks $(|\Theta| - 1)$ more rules than RBAT does in a single split-check iteration. This happens when the first PS-rule in Θ is unprotected. On the other hand, considering a split-check iteration

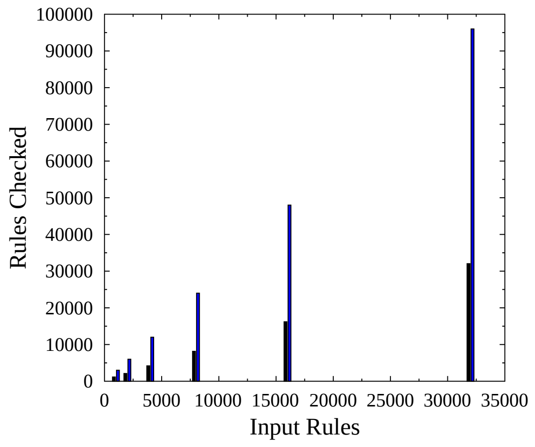
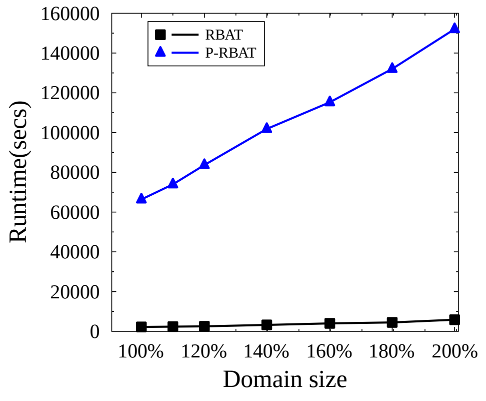


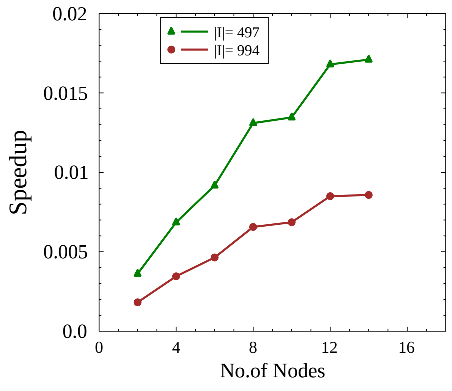
Figure 4.7: Rules Input vs. Checked (k = 60, c = 0.8)

when all PS-rules are protected, both implementations check all $|\Theta|$ rules. In this case, P-RBAT is observed to perform better than RBAT, provided that the size of Θ is large (larger than 2K in our settings). To generalise this, suppose that an anonymization produces a split tree with n nodes (corresponding to split-check iterations the algorithm performs). The number of inner nodes of the split-tree corresponds to the split-check iterations when all rules are protected. So given that the size of the split-tree is n, the number of times RBAT and P-RBAT check exactly the same number of PS-rules is $\lfloor \frac{n}{2} \rfloor$. The remaining $\lceil \frac{n}{2} \rceil$ are leaf nodes and correspond to the split-check iterations when check returns false *i.e.* at least one PS-rule is unprotected. So with n split-check iterations, P-RBAT at maximum checks $(|\Theta| - 1) \times \lceil \frac{n}{2} \rceil$ more rules than the sequential check by RBAT. So the better resource utilization can be expected to be at first $\lfloor \frac{n}{2} \rfloor$ split-check iterations which may be offset by extra communication cost of later $\lceil \frac{n}{2} \rceil$ split-check iterations.

To further investigate this, we set $k = 80$ and $c = 0.7$ and analysed the performance. The first split does not cause any PS-rule not protected. So the difference of the number of rules checked by P-RBAT and the rules checked by RBAT is smaller than when $k = 60$ and $c = 0.8$. This indicates that in order to make the algorithm scalable, the bottleneck due to the difference of the PS-rules checked by RBAT and P-RBAT needs to be addressed.



(a) $|I|$ vs. Runtime



(b) Speedup with respect to $|I|$

Figure 4.8: Performance with respect to $|I|$

We also tested the effect of $|I|$ on the scalability of our method. As shown by Figure 4.8-(a), the runtime of P-RBAT grows much faster than RBAT as $|I|$ increases. 4.8-(b) shows the effect of $|I|$ on scalability of the algorithm with respect to computing resources. The runtime was observed to increase by at least 225%, as we double the domain size. This is because the the number of MapReduce rounds vary with domain size. Setting up a MapReduce round incurs the cost of setting up Map and Reduce tasks, I/O cost of reading input and writing output. Therefore our algorithm makes the effective use of resources when the domain size is small.

We also measured the scaleup [111, 46] which captures the scalability of a parallel algorithm with respect to increasing size of computing cluster and the data size at the same time. We measured the scaleup of P-RBAT by the ratio of the time taken by a cluster size of one computing node only, to the time taken by m nodes on a workload of $(m \times 8)$M transactions. As shown by the results (Figure 4.9), the response time of our parallel method remain nearly constant as we increase data size as well as cluster size. This shows that our algorithm can scalably deal with large problem instances

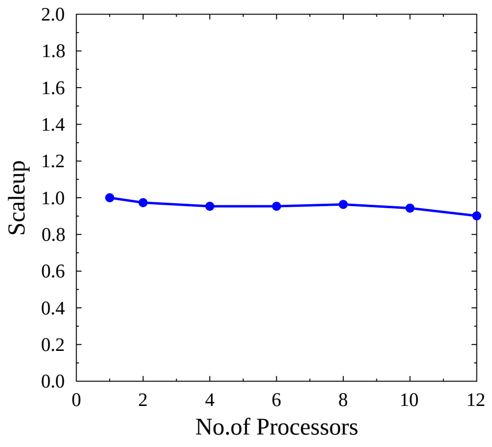


Figure 4.9: Scaleup

as large cluster size becomes available. Throughout the experiment, the scaleup was observed to drop by less than 10%. This is because of ineffective resource utilization due to design side restrictions. For example, splitting a generalized item $\tilde{i}$ requires $|\tilde{i}| - 1$ MapReduce rounds. At each MapReduce round, our design restricts the number of machines to be used at the reduce phase to one. Therefore, increasing the number of available processors can increase the number of mappers used in anonymization, causing more intermediate output to be shuffled to the reducer, thereby incurring a high network cost.

4.5 Summary

In this chapter, we attempted to address the problem of achieving scalability in large transaction data anonymization. We proposed a partitioning method which allows to deal with large datasets and the associated domain of items. Our partitioning scheme allows most of the computations to be performed locally and therefore does not incur high communication cost. To keep the tasks independent at each operation, our method replicates a small amount of auxiliary input such as PS-rules or split-cut to each slave node. We also proposed a MapReduce-based parallization of RBAT. Our design preserves the processing logic of RBAT, so gives exactly the same privacy/utility level as RBAT does. Our experimental evaluation demonstrates that our method can deal with large datasets in a scalable way. But the response time of our algorithm is unreasonably slow. This is because of the parallelism overhead due to iterative computations at the split phase and the extra computational cost due to the lack of interaction among processing nodes at the check phase. In the next chapter, we will attempt to address these performance bottlenecks at the split and check phases.

Chapter 5

MR-RBAT: A Scalable Transaction Data Anonymization Method

In the previous chapter, we discussed the direct parallel implementation of RBAT which partitions data and performs different operations of transaction data anonymization in parallel. The experimental evaluation showed that the algorithm can handle increasingly large volumes of data in a scalable way. The algorithm preserves the processing logic of RBAT, and therefore guarantees the same level of privacy and data utility as RBAT does. But its response time is unacceptably slow. We attribute this to the limitations of MapReduce. Our solution is iterative and each iteration requires a number of MapReduce rounds. MapReduce does not support iterative processing well [62, 9]. In MapReduce, each round is configured separately. This generates a significant overhead which offsets the gains from parallel processing. Also, MapReduce does not allow communication among processing nodes at the Map and Reduce stages. This causes the algorithm to perform extra computation at each iteration. In this chapter, we give a detailed analysis of the performance bottlenecks of our parallel solution and present a solution to address them. We also give an analytical study to estimate the performance of our method based on given parameter settings. Section 5.1 proposes improvements to our parallel anonymization method (described in Section 4.3) to deal with performance bottlenecks. We give our analytical study to estimate the performance of our solution in Section 5.2. Section 5.3 empirically evaluates our method and finally Section 5.4 concludes the chapter.

5.1 MR-RBAT

In this section, we address the performance bottlenecks of our direct parallel method. Recall that given a generalized item $\tilde{i}$, the split phase takes $(\tilde{i}-1)$ MapReduce rounds to find two more specialized items of $\tilde{i}$. This incurs large amount of parallelism overhead, especially at the early splits of the anonymization process. For example, at the first split, $\tilde{i} = P$ therefore the first split causes $(|P| - 1)$ MapReduce rounds. Assuming that $\tilde{i}$ is split into $\tilde{i}_l$ and $\tilde{i}_r$, each specialized item containing $\frac{|P|}{2}$ items. Therefore, splitting $\tilde{i}_l$ and $\tilde{i}_r$ requires $(\frac{|P|}{2} - 1)$ MapReduce rounds. Comparing to the first split, these two splits incur less parallelism overhead but it may still be large enough to offset the performance gains from parallelism. The algorithm can be expected to gain performance after the first few splits. But this may not always be the case. For example, splitting $\tilde{i}$ may result in split with $|\tilde{i}_l| = 1$ and $|\tilde{i}_r| = (|\tilde{i}| - 1)$. In such cases, the later splits may also be expensive in terms of parallelism overhead and may cause the overall performance gain to be offset by parallelism overhead incurred by splitting generalized items.

The extra computational cost incurred by the rule checking in parallel also needs to be addressed. Since rule checking in parallel is performed using a single MapReduce round, it does not incur high cost of setting up mappers/reducers and reading data. But using a single MapReduce round results in limited communication among processing nodes. As a result, all PS-rules are checked by the parallel method. During the early split-check iterations, no PS-rule is unprotected and hence causes the parallel method to perform better than its sequential counterpart. But when any PS-rule is unprotected, the number of PS-rules checked by the parallel method may be greater than those checked by its sequential counterpart and hence may result in extra computational as well as communication cost.

Our proposed method address these performance bottlenecks by controlling the number of MapReduce rounds at the split phase and by employing more than one MapReduce round at the check phase. This will reduce the parallelism overhead at the split phase and will increase the communication and synchronization at the check phase. Fixing the number of MapReduce rounds used at each stage will fix the associated parallelism overhead and may result in better performance of the algorithm. But setting them to some arbitrary value may affect data utility or may not result in effective resource utilization in all problem instances. We allow the user to fix the number of MapReduce rounds to be employed at the split and check phases.

Algorithm 5.1 shows the overall structure of MR-RBAT. Algorithm 5.1 itself is performed sequentially (by a single computer), and specifies the different MapReduce operations, their dependencies and interaction with other sequential and/or parallel operations. Note that all the MapReduce operations use the same data partitioning and representation mechanisms we discussed previously in Section 4.1.

The algorithm starts with a pre-processing phase described in Section 4.3 and follows the same top-down specialization as RBAT. The update is also performed in the same way discussed in Section 4.3. The split and check phases are performed in a more generalized way than our direct parallel implementation (discussed in Chapter 4), so that the overhead can be controlled. In following, we describe in detail the revised design of split and check phases.

5.1.1 α-Split

Recall that RBAT finds specialization of a generalized item using a two-step heuristic. Given a generalized item $\tilde{i}$ to split, the first step uses a single MapReduce round to find a pair of items in $\tilde{i}$ with maximum utility loss when generalized together. The

Algorithm 5.1: MR-RBAT (D, Θ, $\tilde{i}$, k, c)

Input: Original dataset D, a set of PS-rules Θ, the most generalized item $\tilde{i}$, minimum support k and maximum confidence c.

Output: Anonymized dataset $\tilde{D}$

```
1:  $\mathcal{P} \leftarrow$ ComputeUL($P, D$)
2:  $\tilde{P} \leftarrow \{\tilde{i}\}$
3:  $Q$.enqueue($\tilde{i}$)
4:  **while** $|Q| > 0$ **do**
5:      $\tilde{i} \leftarrow Q$.dequeue()
6:      $\{\tilde{i_l}, \tilde{i_r}\} \leftarrow \alpha$-Split ($\tilde{i}$, $\tilde{P}$, $\tilde{D}$)
7:      $\tilde{P}' \leftarrow$ UpdateCut($\tilde{P}$, $\tilde{i}$, $\tilde{i_l}$, $\tilde{i_r}$)
8:      **if** $\gamma$-Check ($D$, $\Theta$, $\tilde{P}'$, $k$, $c$) = true **then**
9:          $Q$.enqueue($\tilde{i_l}$)
10:         $Q$.enqueue($\tilde{i_r}$)
11:         $\tilde{P} \leftarrow \tilde{P}'$
12:     **end if**
13: **end while**
14: $\tilde{D} \leftarrow$ Generalize ($D, \tilde{P}$)
15: **return** $\tilde{D}$
```

second step is sequential and uses $\langle i_x, i_y\rangle$ to split $\tilde{i}$ into two less generalized items $\tilde{i_l}$ and $\tilde{i_r}$. RBAT does this by assigning i_x and i_y to I_l and I_r first, then considering each item $i_q \in \tilde{i} - \{i_x, i_y\}$ in turn and assigning it to either I_l or I_r based on $UL(I_l \cup i_q)$ and $UL(I_r \cup i_q)$. The direct parallelization of this heuristic (discussed in the previous chapter) requires $|\tilde{i}| - 2$ MapReduce rounds, as the assignment of each item is recursively dependent on the assignment of the items preceding it. In the worse case when the most generalized item is split to single items, one per iteration, it will require a total of O($|P|^2$) MapReduce rounds. This will result in a significant setup and data loading overhead.

Alternatively, one may split $\tilde{i}$ based on seeds only. That is, we decide whether an item $i_q \in \tilde{i}$ should be assigned to I_l or I_r based on $UL(i_q, i_x)$ and $UL(i_q, i_y)$. This would then require only a single MapReduce round to split $\tilde{i}$. While this can cut the number of MapReduce rounds significantly, it may cause substantial data utility loss. Consider an extreme case where $\tilde{i} = \{i_1, \ldots, i_{|P|}\}$ is the most generalized item, i_1 and $i_{|P|}$ are the seeds, $\sigma_D(i_j) < k/4, j < |P|$ and $k/2 < \sigma_D(i_{|P|}) < k$. Assuming that a uniform weight of 1 is used in UL calculation, then it is easy to see that using this strategy all the items will be generalized with i_1, resulting in $\tilde{i}_l = (i_1, \cdots, i_{(|P|-1)})$ and $\tilde{i}_r = (i_{|P|})$. As $\sigma_D(i_{|P|}) < k$, $\tilde{i}$ cannot be split, and the data has to be generalized using the most generalized item $\tilde{i}$, incurring a substantial utility loss.

Observing that the split by seeds approach and RBAT are two extremes of parallelization, we propose a generalization which covers a range of solutions between the two extremes. More generally, we specify the upper bound α of MapReduce rounds to employ for each split of MR-RBAT.

Definition 6 (α-Split)**.** *Given a generalized item $\tilde{i}$ and a pair of seeds i_x and i_y, α-*Split*, $1 \leq \alpha \leq |\tilde{i}| - 2$, partitions $\tilde{i}$ into α buckets and splits $\tilde{i}$ in α iterations. Items in each bucket are split based on seeds only, and the splits obtained from the previous iterations are used as the seeds in the current iteration.*

We revised the split phase of our direct parallel anonymization method (discussed in previous chapter) using Definition 6 and call it α-Split. The pseudocode of α-Split is given in Algorithm 5.2. The algorithm first finds the pair of items with maximum UL. The search of the maximum pair is performed, using the same Max_Pair method described in Section 4.3. The pair is then used to initialize I_l and I_r (step 2), the two subsets containing items of specializations of $\tilde{i}$. All the remaining items of $\tilde{i}$ are partitioned into α buckets (step 3). Currently, the partitions are created by sequentially assigning an almost equal number $b = \frac{|\tilde{i}|-2}{\alpha}$ of items to the partitions. Let $i' = \tilde{i} - \langle i_x, i_y \rangle$, the partitions are made so that first partition of $\tilde{i}$ contains $\{i'_1, \ldots, i'_b\}$, the second partition contains $\{i'_{(b+1)}, \ldots, i'_{2b}\}$, and so on.

Algorithm 5.2: α-SPLIT ($\tilde{i}$, α)

Input: A generalized item $\tilde{i}$ to split, and split threshold α.

Output: Less generalized items $\tilde{i}_l$ and $\tilde{i}_r$

```
1:  $\{i_x, i_y\} \leftarrow$ FINDMAXPAIR ($\tilde{i}$)
2:  $I_l \leftarrow i_x, I_r \leftarrow i_y$
3:  $\{\tilde{i}_1, \tilde{i}_2, \ldots, \tilde{i}_\alpha\} \leftarrow$ PARTITION($\tilde{i} - \{i_x, i_y\}$)
4:  for $h = 1$ to $\alpha$ do
5:      Map($m_i$, $\langle I_l, I_r, \tilde{i}_h, D_m \rangle$)
6:        $\mathcal{D}_m \leftarrow$ Load the $m$-th partition of $D$ from DFS
7:        for each $i_q \in \tilde{i}_h$ do
8:          $\tilde{i}_l \leftarrow I_l \cup \{i_q\}, \tilde{i}_r \leftarrow I_r \cup \{i_q\}$
9:          EMIT($i_q, \langle \sigma_{D_m}(\tilde{i}_l), \sigma_{D_m}(\tilde{i}_r) \rangle$)
10:       end for
11:     Reduce($i_q$, $[\langle \sigma_{D_1}(\tilde{i}_l), \sigma_{D_1}(\tilde{i}_r) \rangle, \langle \sigma_{D_2}(\tilde{i}_l), \sigma_{D_2}(\tilde{i}_r) \rangle, \ldots, \langle \sigma_{D_M}(\tilde{i}_l), \sigma_{D_M}(\tilde{i}_r) \rangle]$)
12:       if $UL(\tilde{i}_l) > UL(\tilde{i}_r)$ then
13:         $I_l \leftarrow I_l \cup \{i_q\}$
14:       else
15:         $I_r \leftarrow I_r \cup \{i_q\}$
16:       end if
17:       EMIT($r, \langle I_l, I_r \rangle$)
18: return $\langle \tilde{i}_l = I_l, \tilde{i}_r = I_r \rangle$
```

Given $I_l = i_x$ and $I_l = i_y$, α-Split takes α MapReduce rounds to split $\tilde{i}$ into two less generalized items. At the j_{th} round, the algorithm takes a partition $\tilde{i}_j$ ($1 \leq j \leq \alpha$) and decides for each item $i_q \in \tilde{i}_j$ whether to generalize it with I_l or I_r. Each mapper reads a copy of I_l, I_r, $\tilde{i}_j$ and its assigned subset of D. It then iterates over each item $i_q \in \tilde{i}_j$ and computes the partial support of $gen(I_l \cup i_q)$ and $gen(I_r \cup i_q)$ locally (steps 5-10). All partial supports are then shuffled to the reducers. Each reducer aggregates all partial supports related to i_q and generalizes i_q with I_l or I_r, based on which generalization incurs less UL (steps 11-18). I_l and I_r are then sequentially updated to contain all the items of $\tilde{i}_j$ (step 19). This fixes the number of MapReduce rounds and hence the associated parallelism overhead incurred by the split phase does not vary with the size of the generalized item considered.

Note that the partitions are processed in sequence and the subsets I_l and I_r are updated at the end of each iteration. Also, as the items are assigned to the partitions (we alternatively use the term buckets) in the same order as they appear in $\tilde{i}$, we have the following Lemma.

Lemma 1. *The split stage of our direct parallel anonymization (MR-Split) is a special case of α-Split.*

Proof. Let $\tilde{i} = (i_1, \ldots, i_{|\tilde{i}|})$ be a generalized item to split with the pair $\langle i_x, i_y \rangle$ incurring maximum UL. Dividing the items in $\tilde{i}' = \tilde{i} - \langle i_x, i_y \rangle$ into α partitions gives $\{\tilde{i}_1, \ldots \tilde{i}_\alpha\}$. The partitions are processed sequentially. At the j_{th} MapReduce round, the subsets I_l and I_r contain the items of all preceding partitions *i.e.* all the items of $\{\tilde{i}_1, \ldots, \tilde{i}_{(j-1)}\}$. Setting $\alpha = |\tilde{i}| - 2$, the size of each partition will be $\frac{|\tilde{i}-2|}{\alpha} = 1$. Consider that the items are assigned to the partitions in the same sequence as they appear in $\tilde{i}$, we have $\tilde{i}_1 = i'_1, \ldots \tilde{i}_\alpha = i_{|\tilde{i}'|}$ in which case each partition contains an individual item. At the j_{th} MapReduce round, the algorithm generalizes i'_j to either I_l or I_r and the subsets I_l and I_r contain $\{i_1, \ldots, i_{(j-1)}\}$ and hence gives the same specialized items as MR-Split. □

Lemma 1 shows that α-Split can be used to produce the same specializations of $\tilde{i}$ as produced by RBAT. But other anonymized solutions can also be produced using different settings of parameter α.

Example 4. *Consider a generalized item $\tilde{i} = (a, b, c, d, e, f)$ to split. Suppose the pair of items of $\tilde{i}$ incurring maximum UL when generalized together is $\langle a, e \rangle$ which gives $\tilde{i}' = \tilde{i} - \langle a, e \rangle$. Let $\alpha = 2$ and the number of mappers is set to 2. Let $\tilde{i}'$ be partitioned into $\tilde{i}_1 = \{b, c\}$ and $\tilde{i}_2 = \{d, f\}$.*

During the first MapReduce round, $I_l = a$ and $I_r = e$. Each mapper m gets a data partition D_m of size $\frac{|D|}{2}$ and the first bucket $\tilde{i}_1 = \{b, c\}$, and computes the supports $\sigma_{D_m}(a \cup b)$, $\sigma_{D_m}(e \cup b)$, $\sigma_{D_m}(a \cup c)$ and $\sigma_{D_m}(e \cup c)$. Suppose that the number of reducers is set to 2. All the partial supports corresponding to the item b of the bucket go to the first reducer and those corresponding to c are sent to the other reducer, These partial supports are then aggregated by the reducers.

The first reducer computes and compares UL(gen(a, b)) and UL(gen(b, e)) and generalizes b with $I_l = a$ and the second reducer computes UL(gen(a, c)), UL(gen(c, e)) and generalizes c with $I_r = e$. I_l and I_r are then updated to contain items from $\tilde{i}_1$ before moving on to $\tilde{i}_2$. The next MapReduce round repeats the same process over $\tilde{i}_2 = \{d, f\}$. Given that $I_l = \{a, b\}$ and $I_r = \{c, e\}$, the mappers compute $\sigma_{D_m}(a \cup b \cup d)$, $\sigma_{D_m}(a \cup b \cup f)$, $\sigma_{D_m}(c \cup e \cup d)$ and $\sigma_{D_m}(c \cup e \cup f)$. These partial supports are used by the reducers to compute UL of all the pairs. Suppose that $UL(d \cup I_l) > d \cup I_r)$ and $UL(f \cup I_l) > UL(f \cup I_r)$, we have $\tilde{i}_l = (a, b)$ and $\tilde{i}_r = (c, d, e, f)$ as two less generalized representations of $\tilde{i}$.

5.1.2 γ-Check

Recall that the direct parallel implementation of RBAT (discussed in the previous chapter) checks all PS-rules in one MapReduce round. On the one hand, this incurs a small parallelism overhead which does not vary with the size of Θ. The algorithm will also require reading the dataset only once in parallel and hence the I/O cost will not be large either. But this also limits the communication that can be made among mappers. MapReduce is mainly designed for batch-processing over a shared-nothing computing cluster. During a MapReduce round, the only way to perform communication and synchronization among worker nodes is by using the shuffle phase. It does not allow determination of unprotected rules (if any) until the whole round is finished. For example, if $p \to s \in \Theta$ is the first PS-rule and is unprotected, the algorithm can not determine this at the map stage since there is no communication among mappers except to reducers by shuffling partial support for aggregation. So the algorithm will check all PS-rules in each check phase, even when the first PS-rule is unprotected. This will incur the extra computational cost of processing all PS-rules, and also increase the network cost because of shuffling partial supports of Θ rules to reducers.

Alternatively, the rule checking can be performed in the same way as RBAT does (Section 3.2). This will require O($|\Theta|$) MapReduce rounds. At each round, the algorithm takes a PS-rule and checks for its protection. At the end of each MapReduce round, the algorithm synchronizes the results of the PS-rules checked so far and decides whether to check the remaining PS-rules. This will allow the algorithm to stop as soon as the first unprotected (if any) PS-rule is found but will also incur the I/O and setup cost proportional to $|\Theta|$.

Observe that sequential rule checking and checking rules in one MapReduce round are two extremes of tradeoff between parallelism overhead and the extra computational cost. Sequential rule checking will check the same number of rules checked by RBAT so does not incur any extra computational cost but it will incur parallelism overhead proportional to the size of Θ. Performing rule checking using only one MapReduce round will have the minimum parallelism cost but it will incur maximum computational cost. To allow a balance between parallelism overhead and the extra computational cost, we propose a more general approach which allows the number of MapReduce rounds to be specified and controlled.

Definition 7 (γ-Check)**.** *Given a set of PS-rules* Θ*,* γ-Check*,* $1 \leq \gamma \leq |\Theta|$*, checks* Θ *in* γ *iterations. Each iteration checks* $\frac{|\Theta|}{\gamma}$ *PS-rules in* Θ*.*

Using Definition 7, we present the revised version of the check stage of our direct parallel method (Section 4.3) in Algorithm 5.3. The algorithm partitions Θ into γ subsets (step 1) and checks the subsets in sequence. The partitions are made in a way that each subset contains $\frac{|\Theta|}{\gamma} = n$ consecutive PS-rules of Θ. That is, $\Theta_1 = \{p_1 \rightarrow s_1, \ldots, p_n \rightarrow s_n\}$, $\Theta_2 = \{p_{(n+1)} \rightarrow s_{(n+1)}, \ldots, p_{2n} \rightarrow s_{2n}\}$ and so on. During the j_{th} MapReduce round, a mapper m loads its assigned data partition D_m (step 4) and reads a copy of Θ_j. It then iterates over each rule $p \to s \in \Theta_j$, computing the partial support of $\tilde{p}$ and $\tilde{p} \cup s$ (steps 5-8), where $\phi(p) = \tilde{p}$ generalizes p. Note that the support computation is performed using the original dataset D and the current split cut $\tilde{P}$, and in the same way as the check stage of our direct parallel method (MR-Check) does.

The partial supports from all mappers with the associated PS-rules as their keys are then shuffled to reducers. A reducer aggregates all partial supports pertaining to a PS-rule and checks the protection conditions specified in Definition 3 (steps 9-13). The algorithm will not undergo any subsequent rounds, if any rule is found unprotected in the current round. Hence, γ-Check at maximum checks $(\frac{|\Theta|}{\gamma} - 1)$ more rules than RBAT does, but would require a maximum of γ MapReduce rounds only.

Example 5. *Consider checking* $\Theta = \{b\ e \to g, a \to j, c\ d \to h, f \to l\}$ *using the dataset shown in Table 3.1. Let* $k = 2$*,* $c = 0.4$ *and* $\gamma = 2$*. The first MapReduce will check first* $\frac{|\Theta|}{\gamma} = 2$ *PS-rules. Let* D *is divided into two partitions with* D_1 *containing first three and* D_2 *containing last two transaction records. The first mapper computes the partial support from the first data partition i.e.* $\sigma_{D_1}(b\ e) = 2$*,* $\sigma_{D_1}(b\ e \cup g) = 1$*,* $\sigma_{D_1}(a) = 3$*, and* $\sigma_{D_1}(a\ j) = 1$*. The second mapper computes the partial support from* D_2 *i.e.* $\sigma_{D_2}(b\ e) = 2$*,* $\sigma_{D_2}(b\ e \cup g) = 1$*,* $\sigma_{D_2}(a) = 2$*, and* $\sigma_{D_1}(a\ j) = 0$*. Assume that* $R = 2$*, the first reducer aggregates the partial support associated with* $b\ e \to g$ *and returns false since* $Conf(b\ e \to g) = 0.5$ *and the second reducer returns true since* $Conf(a \to j) = 0.2$*. The algorithm will not undergo the next MapReduce to check any further PS-rules. The first PS-rule checked by RBAT is* $b\ e \to g$ *so our parallel solution incurs the overhead of checking* $(\frac{|\Theta|}{\gamma} - 1) = 1$ *extra PS-rule.*

Performing rule-checking in some constant number of rounds provides a way to limit the parallelism overhead to some constant. This also allows to control the trade off between the cost of performing extra computations and the parallelism overhead. That is, reducing the number of MapReduce rounds to employ at the rule checking phase allows more computations to be performed at each MapReduce round and reduces the synchronization level at the rule checking phase. It also allows to control the amount of

Algorithm 5.3: γ**-CHECK** (D, $\tilde{P}$ Θ, k, c, γ)

Input: Original dataset D, Split-cut $\tilde{P}$, PS-rules Θ, Minimum support k, Maximum confidence c, Check threshold γ.

Output: True if all PS-rules of Θ are protected and False otherwise.

```
1: $\{\Theta_1, \Theta_2, \ldots, \Theta_\gamma\} \leftarrow$ PARTITION($\Theta$)
2: $j \leftarrow 1, protected \leftarrow true$
3: while $j \leq \gamma$ and $protected = true$ do
4:    Map($m$, $\langle D_m$, $\Theta_j \rangle$)
5:       $\mathcal{D}_m \leftarrow$ Load the $m$-th partition of $D$ from DFS
6:       for each rule $p \rightarrow s \in \Theta_j$ do
7:          $\tilde{p} \leftarrow \phi(p)$
8:          COMPUTE AND EMIT ($p \rightarrow s, \langle \sigma_{D_m}(\tilde{p}), \sigma_{D_m}(\tilde{p} \cup s) \rangle$) using Observation 1
9:       end for
10:   Reduce($p \rightarrow s, [\langle \sigma_{D_j}(\tilde{p}), \sigma_{D_j}(\tilde{p} \cup s) \rangle, 1 \leq j \leq M]$)
11:      $\sigma_D(\tilde{p}) = \Sigma_{j=1}^{M} \sigma_{D_j}(\tilde{p}),\ \ \sigma_D(\tilde{p} \cup s) = \Sigma_{j=1}^{M} \sigma_{D_j}(\tilde{p} \cup s)$
12:      if $\sigma_D(\tilde{p}) < k$ or $\frac{\sigma_D(\tilde{p} \cup s)}{\sigma_D(\tilde{p})} > c$ then
13:         EMIT($r$, False)
14:      end if
15: update $protected \leftarrow false$, if any reducer emits False.
```

memory required at each mapper and reducer in a MapReduce round. That is, using a small number of MapReduce rounds causes the increased amount of memory required to hold the map output and the reduce input at each MapReduce round and increasing the number of MapReduce to be employed at the rule checking phase will produce a large number of map output pairs to be shuffled to reducers and processed by reducers at the reduce phase.

Assuming that PS-rules are partitioned in the same sequence as they appear in Θ and the subsets are checked in sequence, we have Lemma 2.

Lemma 2. *MR-Check and the check phase of RBAT are special cases of γ-Check.*

Proof. Let $|\Theta|$ be the number of rules to be checked. Setting $\gamma = 1$ gives the number of PS-rules to be checked at each MapReduce round as $\frac{|\Theta|}{\gamma} = |\Theta|$. In this case, all the PS-rules are checked in one round only and hence emulates MR-Check. Similarly, setting $\gamma = |\Theta|$ gives $\frac{|\Theta|}{\gamma} = 1$ PS-rule to be checked in a MapReduce round, hence emulates the sequential rule-checking of RBAT. □

Note that each MapReduce round requires to read D into the memory of worker nodes, therefore setting γ to large values may cause a significant I/O cost, especially when $|D|$ is large. We will further discuss this in Section 5.2.

From Lemmas 1 and 2, we have Corollary 3.

Corollary 3. *MR-RBAT is a more general form of RBAT.*

Proof. Consider a generalized item $\tilde{i}$ is to be split. Consider that the pair of items with maximum UL is chosen as seeds to initialize I_l and I_r and let α be set to $(|\tilde{i}| - 2)$. Following Lemma 1, α-Split gives the same specializations as the split phase of RBAT does. Similarly, let the number of rules to be checked be $|\Theta|$. Let $\gamma = |\Theta|$, and following Lemma 2, the algorithm checks the same PS-rules checked by RBAT. Setting $\alpha = (|\tilde{i}| - 2)$ and $\gamma = |\Theta|$ gives a special case of MR-RBAT. Since α and γ be set to any values up to $(|\tilde{i}| - 2)$ and $|\Theta|$ respectively, MR-RBAT is a more generalized form of RBAT. □

5.2 Performance Estimation

This section presents an analytical study on the performance of MR-RBAT. We analyse the cost of MR-RBAT's split and check phases in terms of data and hardware characteristics. More specifically, we attempt to find the cost of split and check phases with respect to datasize, size of a generalized item considered for a split, the number of PS-rules, the number of mappers M and reducers R. For the sake of simplicity we assume that we have $(M + R)$ number of physical computing cores to run all M mappers and R reducers in parallel and the number of map tasks are also the same as the number of mappers.

The effect of α

In this section, we analyse the cost of α-Split with respect to given parameters. Following the common assumption [38], let t_M, t_S and t_R be the cost of Map, Shuffle and Reduce phases respectively of a single MapReduce round. The cost of finding the specialized representation of a generalized item $\tilde{i}$ is given as follows.

$$T_\alpha = \alpha \times (t_M + t_S + t_R) \tag{5.1}$$

The parallelization overhead of the map phase is constituted by the startup and disk I/O cost. The disk I/O is performed while reading a data partition into the memory of each mapper or spilling the intermediate output to the disk. We ignore the spilling cost and assume that each mapper has enough space to hold the output pairs. Let $s_m(M)$ be the setup cost of M mappers. Given data $|D|$ is read by M mappers in parallel and let ω be the average time that it takes to read a transaction (of average size in D) from the distributed file system, the overall parallelism cost by map phase is given in Equation 5.2

$$t_M = s_m(M) + \frac{|D|}{M} \cdot \omega \tag{5.2}$$

Given α, the map output pairs with the number of distinct keys produced is $\frac{(|\tilde{i}|-2)}{\alpha}$. Given R reducers and considering that the map output pairs with the same key must be sent to a single reducer, the degree of parallelism at the shuffle stage is determined by $min(R, \frac{(|\tilde{i}|-2)}{\alpha})$. Let ξ be the network efficiency constant.

$$t_S = \frac{|\tilde{i}| - 2}{\alpha \times min(R, \frac{|\tilde{i}|-2}{\alpha})} \cdot \xi \tag{5.3}$$

During a MapReduce round, the number of pairs sent by each mapper is $\frac{|\tilde{i}|-2}{\alpha}$. So the number of pairs sent by M mappers will be $M \times \frac{|\tilde{i}|-2}{\alpha}$. Let $s(R)$ be the start up cost of R reducers. $min(R, \frac{|\tilde{i}|-2}{\alpha})$ reducers read $M \times \sqrt{|P'|}$ map intermediate output pairs in parallel, the overall cost of reduce phase is as given in Equation 5.4.

$$t_R = s_r(R) + \frac{M \times (|\tilde{i}| - 2)}{\alpha \cdot min(R, \frac{|\tilde{i}|-2}{\alpha})} \cdot \omega \tag{5.4}$$

Substituting the cost of t_M, t_S and t_R in Equation 5.1, we have the overall cost of α-SPLIT using α iterations to split $\tilde{i}$.

$$T_\alpha = \alpha \times \left[s_m(M) + s_r(R) + \frac{|D|}{M} \cdot \omega + \frac{|\tilde{i}| - 2}{\alpha \times min(R, \frac{|\tilde{i}|-2}{\alpha})} \cdot (\xi + M \cdot \omega)\right] \tag{5.5}$$

Clearly, a large α, which a direct parallelization of RBAT would imply, can result in a significant overhead cost due to the setup and data loading requirements. We will show in Section 5.3 that it is possible to use a small α in split to control overhead while retaining good data utility. Note that at this stage, the number of computations performed (effective CPU utilization) during the split stage is $(\frac{|D|}{M} \times (|\tilde{i}| - 2))$ and does not vary with α.

The effect of γ

This section analyses the cost of γ-Check. Following Equation 5.1, the overhead incurred by a single MapReduce round can be calculated as the sum of parallelism cost at map, shuffle and reduce stages.

The cost incurred by the map phase is computed using Equation 5.2. Given γ MapReduce rounds, a mapper outputs the partial support of $\frac{|\Theta|}{\gamma}$ PS-Rules in a single MapReduce round. The number of parallel connections to be utilized during the shuffle phase is $min(R, \frac{|\Theta|}{\gamma})$. The shuffle cost of a single MapReduce round is given as follows.

$$t_S = \frac{|\Theta|}{\gamma \times min(R, \frac{|\Theta|}{\gamma})} \cdot \xi \tag{5.6}$$

The total number of intermediate pairs produced at the map stage is $M \times \frac{|\Theta|}{\gamma}$ and the number of reducers to be utilized for reading map output is $min(R, \frac{|\Theta|}{\gamma})$, the cost incurred by the reduce phase is given as follows.

$$t_R = s_r(R) + \frac{M \times |\Theta|}{\gamma \cdot min(R, \frac{|\Theta|}{\gamma})} \cdot \omega \tag{5.7}$$

From Equations 5.2, 5.6 and 5.7, the overall parallelism overhead of a γ MapReduce rounds is given as follows.

$$T_\gamma = \gamma \times \big[s_m(M) + s_r(R) + \frac{|D|}{M} \cdot \omega + \frac{|\Theta|}{\gamma \times min(R, \frac{|\Theta|}{\gamma})} \cdot (\xi + M \cdot \omega)\big] \tag{5.8}$$

It is easy to observe from Equation 5.8 that γ also controls the level of parallelism achieved at the shuffle and reduce stages. For example, setting $\frac{|\Theta|}{\gamma} \geq R$ allows all available reducers being effectively utilized at each MapReduce round. This is because MapReduce forces all the pairs with the same key to be processed by same reducer.

Fixing γ also affects the computational cost of γ-Check. At a rule-checking stage, the PS-rules in Θ may or may not be protected. Therefore, we analyse the computational cost of γ-Check under both conditions. Let r be the average rule size. The case when all PS-Rules are protected, the computational cost is given as follows.

$$C_\gamma = |\Theta| \times \big(r + \frac{M}{min(\frac{|\Theta|}{\gamma}, R)}\big) \tag{5.9}$$

Notice that given that all PS-rules are protected, the computational cost at the map is not affected by γ where as the level of parallelism achieved at the reduce stage depends on γ.

Now, we analyse the case when one or more PS-rules are unprotected. Suppose that n_{th} is the first PS-rule found unprotected. In this case, the sequential method checks the first n rules. When checking rules in γ rounds, suppose this unprotected rule is found after checking all the rules of the first q rounds and x more rules of the $(q+1)_{th}$ round by MR-RBAT so the number of rules checked by the sequential method can be represented as follows.

$$n = q \times \frac{|\Theta|}{\gamma} + x \tag{5.10}$$

Where $q \in [1, \gamma]$, $x \in [0, \frac{|\Theta|}{\gamma} - 1]$ and $q \times \frac{|\Theta|}{\gamma} \leq n < (q+1) \times \frac{|\Theta|}{\gamma}$. The actual number of MR rounds the algorithm undergoes γ' can be calculated as follows.

$$\gamma' = \begin{cases} (q+1) & \text{if } x > 0 \\ q & \text{if } x = 0 \end{cases}$$

So MR-RBAT incurs the cost of checking $(\gamma' \times \frac{|\Theta|}{\gamma} - n)$ more rules than the sequential method. Using the above relation and Equation 5.10, it can be observed that when $x = 0$, the number of PS-rules checked by the sequential method and γ-Check is the same.

5.3 Experimental Evaluation

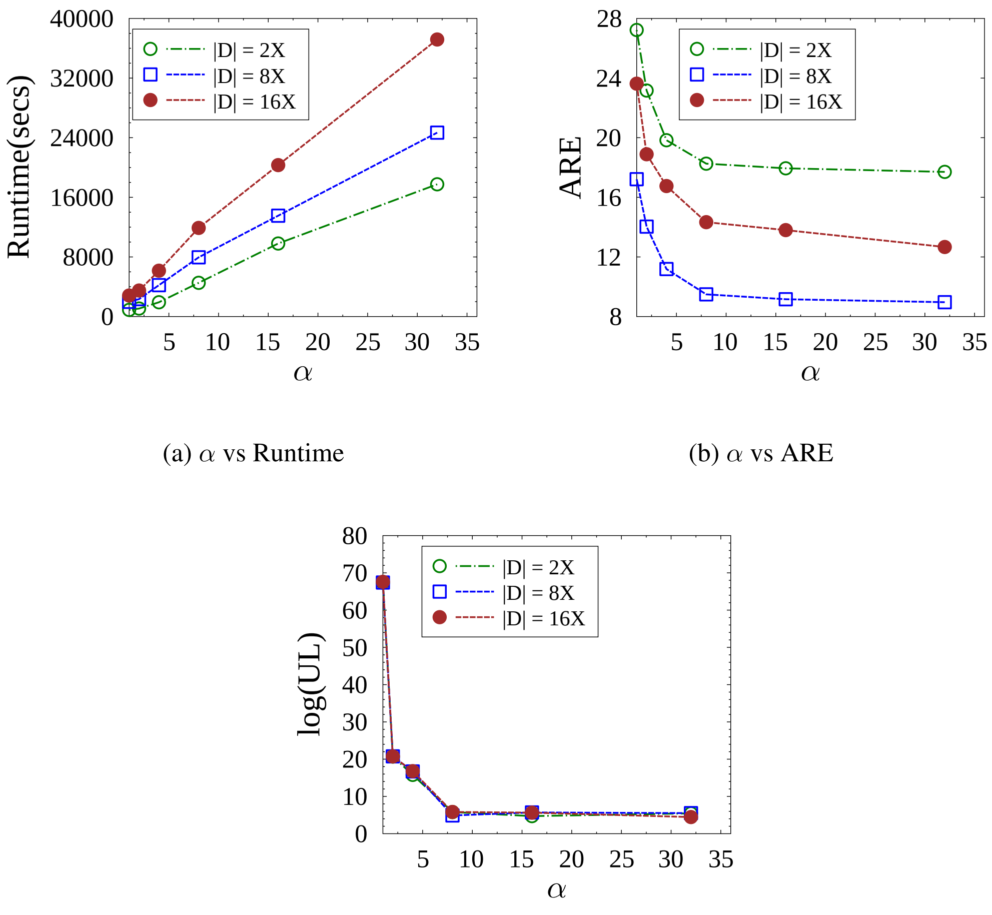
In this Section, we empirically evaluate our work. Section 5.3.1 presents the scalability of our algorithm (discussed in Section 5.1) and the data utility achieved, with respect to different inputs (*e.g.* $|D|$, $|\Theta|$) and the associated parameters (*e.g.* α, γ). Section 5.3.2 empirically tests the cost estimation of MR-RBAT (presented in Section 5.2).

5.3.1 Evaluation of MR-RBAT

We implemented MR-RBAT to run as a master program (Algorithm 5.1). We used ARE (average relative error) [72] and UL measures to quantify the utility of anonymized data. ARE quantifies the data utility by measuring the difference between the support of a query in the original data and the anonymized data. ARE is a widely used application-specific measure, and is independent of the working mechanism of any anonymization method. The UL measure is algorithm-specific, but captures the loss of information without considering any specific application of the anonymized data. We measured ARE using a workload of 1000 randomly generated queries, with an average and maximum size of 6 and 10 items respectively. Unless otherwise specified, we used the same settings described in the previous chapter, except for $|D| = 16M$, $k = 15$ and $c = 0.9$. This caused RBAT to undergo a large number of split-check iterations and hence allowed us to study the effect of γ on data utility.

We first tested the effect of α on runtime and on data utility. As can be seen from Figure 5.1(a), runtime scales largely linearly with respect to α. This suggests that dividing $\tilde{i}$ into different sizes of buckets had little impact on the outcome of split in each iteration. This is further confirmed by the ARE and the UL results in Figure 5.1(b)-(c). When α is very small, many items of $\tilde{i}$ are put into one bucket and assigned to I_l or I_r based on seeds only. This caused items with high ULs to be generalized together. But setting $\alpha > 4$, i.e. buckets are smaller enough, the α value only affected runtime without affecting the utility of anonymized results in a significant way. This shows that the performance of splitting $\tilde{i}$ can be significantly improved by using a relatively small α without compromising data utility.

We also observed an interesting relationship between α and split skewness during these experiments. Let S_α be the split tree constructed by MR-RBAT based on some α, and $\tilde{i}$ be a non-leaf node of S_α with $\tilde{i}_l$ and $\tilde{i}_r$ as its left and right children respectively. We



(a) α vs Runtime

(b) α vs ARE

(c) α vs UL

Figure 5.1: Effect of α

measure split skewness $\xi(S_\alpha)$ using Equation 5.11.

$$\xi(S_\alpha) = \sum_{\tilde{i}\in S_\alpha} \frac{||\, |\tilde{i}_l| - |\tilde{i}_r|\, ||}{|\tilde{i}|} \tag{5.11}$$

It was observed that split skewness decreased as α was increased. This is because when α is small, a large bucket of items will be split in a single round based on the seeds only. If data distribution is such that generalizing most of items in the bucket with one of the seeds produces a larger UL than the other seed does, then most of the items will be generalized with one seed, resulting in a skewed split. Skewed splits are more likely

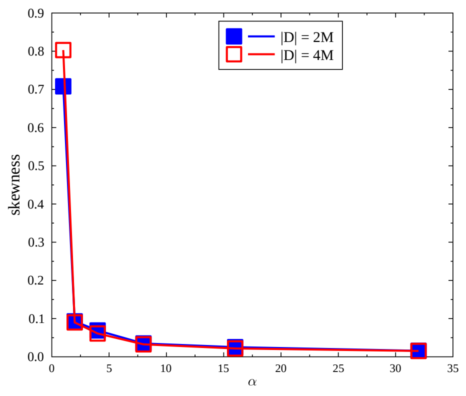
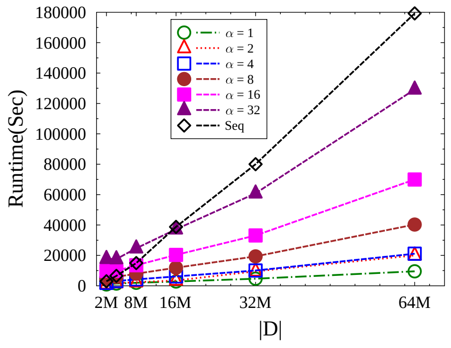


Figure 5.2: α **vs. Skewness**

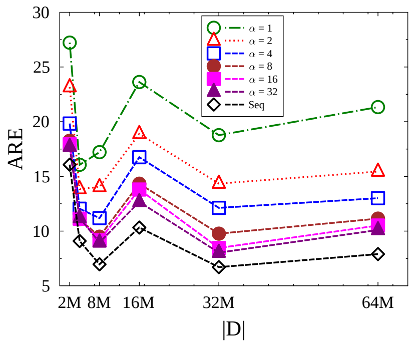
to make Θ unprotected, resulting in an early stop in split and a higher ARE. This is confirmed by Figure 5.2, which shows the average skewness at each split. This further indicates that very small α values are to be avoided.

Figure 5.3 shows how MR-RBAT performed with respect to varying data sizes. For smaller datasets, it performed no better than RBAT due to the overwhelming parallelization overhead. However, the run-time of MR-RBAT grows much more slowly than RBAT does, especially when α is small. This demonstrates its scalability to large datasets. It is interesting to observe the run-time for $\alpha = 32$. At this setting, MR-RBAT and RBAT achieved almost the same performance in terms of runtime. This suggests the severity of overhead caused by iteration using MapReduce: the gains from parallel processing has almost been exhausted entirely by the overhead. Figure 5.3(b) and 5.3(c) show the ARE and UL results in these experiments. It is expected that setting a smaller α would help performance, but could potentially affect the utility of anonymized results. However, Figure 5.3(b)-(c) shows that almost identical utility can be achieved when $\alpha = 32$. This confirms the feasibility and effectiveness of using α in the split phase of MR-RBAT to balance performance and utility retention.

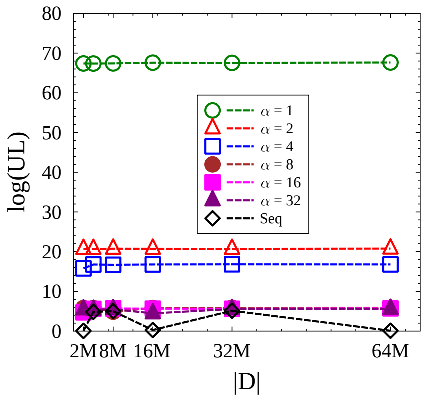
We also evaluated the scalability of MR-RBAT at varying cluster size from 2 to 14,



(a) Runtime vs Data Size



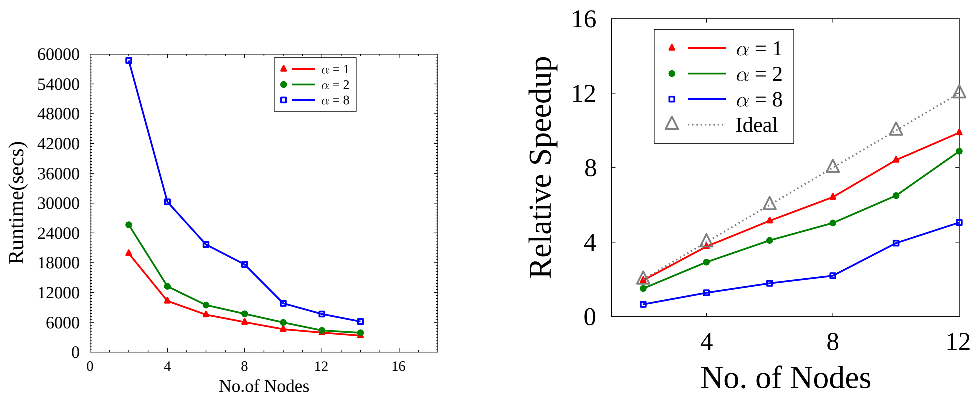
(b) ARE vs Data Size



(c) UL vs Data Size

Figure 5.3: Scalability with respect to Datasize

and at different α values. Figures 5.4(a) and 5.4(b) show the runtime and speedup, respectively. The relative speedup was measured as the ratio of runtime obtained by the sequential RBAT implementation, to the runtime of MR-RBAT with a cluster size whose speedup is measured. Comparing to the direct parallel implementation of RBAT (discussed in the previous chapter), MR-RBAT performs nearly up to eight times better, and hence shows the effectiveness of α towards better resource utilization. It also achieved a near linear speedup when a smaller cluster (relative to data size) was used (up to 4 in our experiments). This is because when a small cluster is used, the workload assigned to each machine is large. As we increase the cluster size, the workload on each machine reduces and causes the performance gain. But after a certain point,

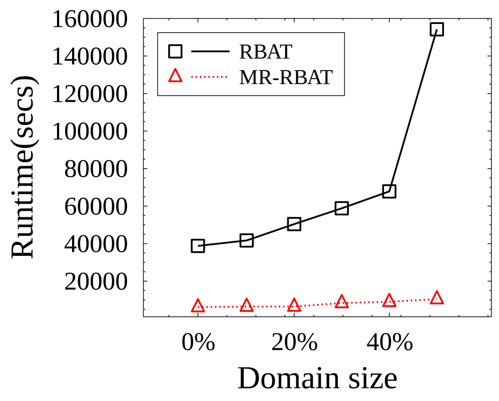


(a) Runtime vs Cluster Size (b) Speedup vs Cluster Size

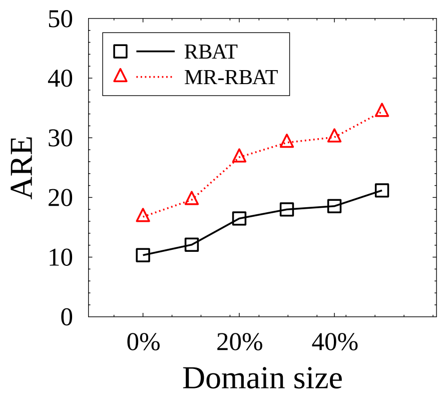
Figure 5.4: Scalability with respect to Cluster Size

the workload on each machine reduces to the level that the setup cost and communication overhead dominates over the actual processing time. For example, using up to 4-nodes cluster sizes, the speedup is nearly linear whereas using a 14-nodes cluster, the speedup drops to nearly $\frac{1}{2}$ of the ideal (linear) speedup. One reason for this ineffective resource utilization is the constant data size of $32M$. When the cluster size increased, the computation performed by each mapper became lighter, and the effect of overhead on runtime became more significant. So adopting a suitable ratio of data size to cluster size is important to achieving a good speedup. We also partially attribute this to the MapReduce platform. MR-RBAT uses one reducer only in the first step of α-SPLIT, and others are used as mappers. Increasing the number of mappers causes more intermediate output pairs to be fetched and processed by the single reducer, thereby degrading the speedup.

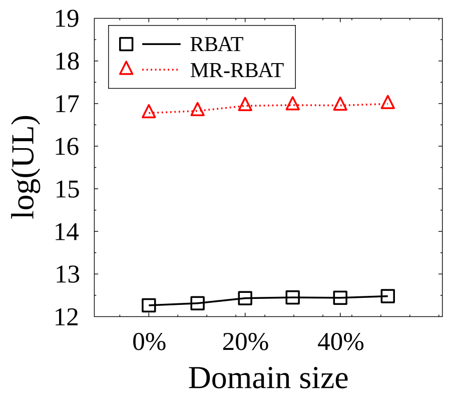
Next, we varied domain size to see its effect on runtime and the data utility. As shown in Figures 5.5, MR-RBAT's runtime was more stable and grew much more slowly than RBAT did as the domain size increased. On the other hand, the difference in ARE as well as in UL between RBAT and MR-RBAT increased slightly as the domain size was increased. This is because MR-RBAT uses a fixed number of MapReduce rounds in split. Increasing domain size causes more items to be put in one bucket. This causes the



(a) Domain size vs Runtime



(b) Domain size vs ARE

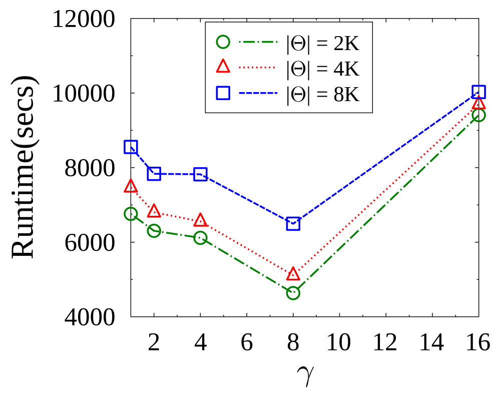


(c) Domain size vs UL

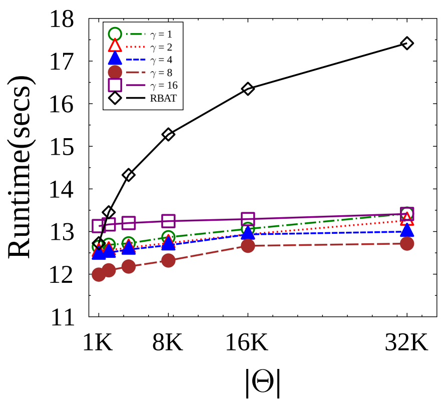
Figure 5.5: Effect of $|I|$

items causing high loss of information to be generalized together. Comparing to direct parallel anonymization (discussed in the previous chapter), the overall performance and the stability of MR-RBAT's runtime growth is higher, but this also contributed to an increased ARE and UL.

We tested the effect of γ on runtime. Note that a γ setting will only affect runtime, not the utility of anonymized results, so only the runtime results are reported in Figure 5.6. Observe that initially the runtime was decreased when we increased γ (see Figure 5.6(a)). This is because when γ is small, the inefficiency introduced by γ-CHECK is mainly from the need to check extra, but unnecessary rules. As γ increases, the number



(a) γ vs Runtime



(b) $|\Theta|$ vs Runtime

Figure 5.6: Effect of γ

of unnecessary rules to check decreases, resulting in a better runtime. However, as we further increase γ, reduction through checking fewer unnecessary rules decreases, but the cost of setting up more MapReduce rounds increases, making an increase in the overall runtime.

Increasing the number of rules also caused the runtime of MR-RBAT to increase, but much more slowly than RBAT did, except for $\gamma = 1$, as shown in Figure 5.6(b). When $\gamma = 1$, MR-RBAT checks every rule in Θ and is not efficient for the reason we gave above. All other settings of γ have resulted better runtime and scalability with respect to the number of rules to be enforced in anonymization.

Finally, we tested the scaleup of our method. As shown by the results in Figure 5.7, the response time of MR-RBAT remains nearly constant as we increase the data size as well as the cluster size. The scaleup was observed to drop by less than 5% and has improved by nearly 5% comparing to our direct parallel implementation. This is because MR-RBAT allows to use more than one reducer at the second step of the split phase whereas the number of reducers at the first step are still restricted to one only. Therefore, the effective resource utilization is not made at the reduce stage.

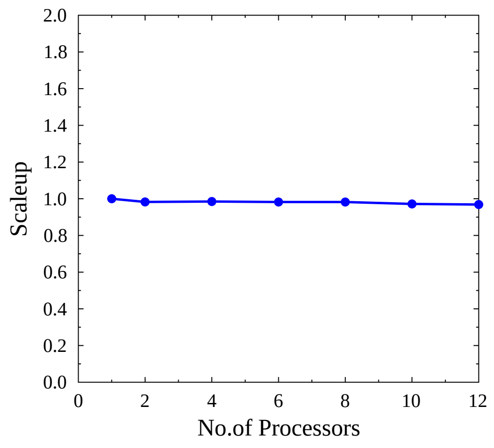


Figure 5.7: Scaleup

Notation	***Empirical Value (Seconds)***
$s_m(M)$	29.99325
$s_r(R)$	15.331
ω	0.00000379
ξ	0.001058298

Table 5.1: Empirical values for evaluation of cost estimation

5.3.2 Evaluation of Performance Estimation

We used our implementation to test the accuracy of our cost estimation study. To carry this out, we first computed the numerical values (shown in Table 5.1) for each variable used in our analytical study and computed the parallelism cost of different settings (1 to 32) of α and γ using Equation 5.5 and 5.8 respectively. The values were computed by running our implementation twenty times and in controlled settings *e.g.* to compute $s_m(M)$, we tune our method to run an empty Map-only job and compute the average of their runtime. The experiments were performed under the same default settings, presented in Table 4.3.

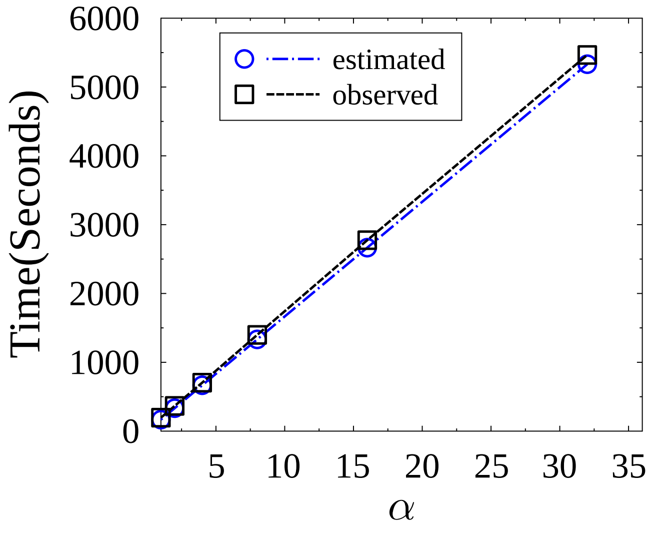


Figure 5.8: Evaluation of cost estimation of α-Split with respect to α

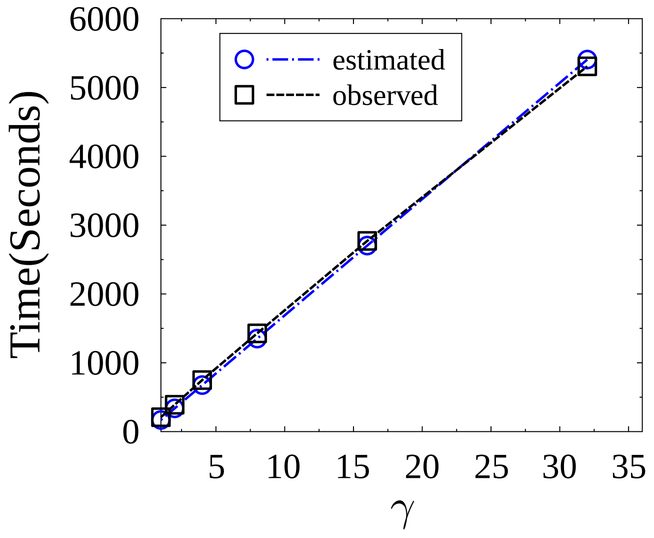


Figure 5.9: Evaluation of cost estimation of γ-Check with respect to γ

To check the accuracy of our analytical study, we compared the estimated cost with the runtime acquired by actually running our implementations of the α-Split (Section 5.1.1) and γ-Check (Section 5.1.2) phases at different α and γ settings respectively.

As shown by our results shown in Figure 5.8 and 5.9, our cost estimation is close to the parallelism cost observed, using the implementations of α-Split and γ-Check. This shows that our analytical study has estimated the cost well, suggesting that relevant parameters have been considered and the model is reasonably accurate. This is significant as this cost model could help set α and γ values, as we have seen in Section 5.2. However, the estimated cost should not be expected to be an exact replication of the actual runtime overhead. We attribute this to the level of details captured in our analytical study. For example, the study does not take into account the overhead which is associated with system calls for opening and reading local files at the map and reduce stages. Note that throughout our experimental study, the number of map output pairs was not high and hence the associated memory requirements were not higher than the memory space available on each physical node running the map instances. Therefore, the actual shuffle time of the α-Split and γ-Check implementations does not vary significantly with varying α and γ settings.

5.4 Summary

In this chapter, we have studied how performance of the direct parallel implementation of RBAT on MapReduce can be improved. The direct parallel anonymization of RBAT using data partitioning can incur an overwhelmingly high parallelization cost due to the iterative operations to be performed. MR-RBAT employs two controls to limit the maximum number of MapReduce rounds to be used during data generalization, thereby reducing the overhead and computational cost. This improves the performance of the algorithm and also causes better resource utilization. We give an analytical study to help estimate parallelism overhead, and hence the performance of our method. Our analysis shows that the parameters α and γ may also be used to control memory requirements in each MapReduce round. This ensures the scalability of our method, when the generalized item to split consists of a large number of items and the number of PS-rules to check is large. We also empirically studied the effect of different settings of these controls and found that MR-RBAT can scale nearly linearly with respect to large problem instances and the increasing cluster size, while retaining good data utility. MR-RBAT also performed nearly up to eight times better than the direct parallel implementation of RBAT, and up to 9 times better than the sequential RBAT implementation over a cluster of 14 machines in our experiments.

Chapter 6

Conclusions and Future Work

This chapter summarises our contributions and discusses possible future research directions based on our current work.

6.1 Research Summary

Transaction data publishing has important applications in different domains, such as personalized medicine and purchasing trend projection. However, identity of the individuals associated with transactions and the sensitive information contained in their transaction records may be disclosed. Set-based generalization methods such as RBAT prevent such attacks while retaining good utility of data, but they are all centralized and can not handle large data volumes in a scalable manner. This research has addressed the problem of scalable transaction data anonymization for large datasets. More specifically, we have parallelized RBAT using MapReduce. The contributions can be summarised as follows:

- **Data Partitioning:** Data Partitioning is important to any algorithm that attempts to process a large amount of data in parallel and it is important to the parallelization of RBAT too. In our work, we proposed a two-way partitioning strategy for transaction data. The importance of our partitioning is two fold. First, our method takes into account the characteristics of transaction data such as the number and the length of transaction records, the number of distinct items in the dataset *etc.* and is independent of the algorithm-specific computations to be performed on the partitions. Our method may be used to perform different operations in other transaction data anonymization algorithms *e.g.* the creation of the count tree to achieve k^m-anonymity [114]. This also allows to balance the workload associated with each partition. Second, our partitioning takes into account the restriction imposed by MapReduce. For example, MapReduce allows limited communication among machines and the machines are only allowed to communicate via network interconnection. Our method creates the partitions in a way that each partition can be processed independently in parallel to other partitions and only a small amount of data needs to be communicated among machines.

- **Parallelization of RBAT:** We proposed two MapReduce-based methods for large-scale data anonymization. Our approaches are the first to address the scalability issues in transaction data anonymization. Our direct parallelization of RBAT (P-RBAT) uses our partitioning method to design the MapReduce-based parallelization of RBAT and allows to handle large datasets in a scalable way. The method preserves the processing logic of RBAT while performing the key operations of RBAT using MapReduce. Therefore, it gives the same level of data utility as RBAT does. The loop-controlled method (MR-RBAT) improves the response time of P-RBAT and allows to deal with the performance bottlenecks which arise due to certain limitations of MapReduce. As suggested by the experiments, the algorithm performs up to 10 times better using a cluster of 14 machines at the data utility loss by 5% comparing to its sequential counterpart. The applicability of our results in terms of scalability, privacy and data utility were also experimentally evaluated on different data, privacy and hardware characteristics and similar results were achieved [82, 81].

Our approaches address the limitations of existing MapReduce-based privacy-preserving approaches outlined in Section 2.4. Both the approaches do not compromise the level of privacy provided by RBAT and therefore prevent the disclosure of both identity and the sensitive information in large transaction data while existing approaches do not prevent both types of disclosure and focus on identity disclosure in relational data.

- α **Control:** MapReduce does not support iterative computation well and a non-negligible performance penalty is paid, since data must be reloaded and reprocessed in each iteration. We proposed α control which allows to control the parallelism overhead by specifying the number of MapReduce rounds to employ for performing iterative computations. Unlike other existing methods (outlined in Section 2.5), our solution does not require any modifications to the framework itself and does not perform any data elimination which may have largely affected the level of privacy and utility that can be achieved in the anonymized data. Our approach also has applications to the parallelization of other iterative methods. For example, COAT [73] protects a privacy constraint by iteratively selecting and generalizing an item from the itemset specified in the privacy constraint that has minimum support in given dataset. A greedy max-k-cover algorithm [52] addresses the problem of covering a maximum set of elements with a fixed number of subsets by iteratively selecting a subset from given input that covers a large number of elements. Different methods for performing operations from large graphs such as multi-way partitioning [60] also requires iterative processing. Using our approach, these methods will allow to control the parallelism overhead.

- γ **Control:** Another limitation of MapReduce is the lack of support of early termination which would allow the processing nodes to cease processing, when a specific condition holds. Such a limitation is due to the result of limited communication among mappers and reducers in a MapReduce round and introduces performance-bottlenecks in various applications. A typical example arises in the context of rank-aware processing *e.g.* performing top-k queries [53] in MapReduce will require to process the whole dataset by mappers and will require to communicate the locally found top-k queries to a single processing node. We proposed γ control that allows to perform the intended processing in multiple phases. At each phase, our approach uses a single MapReduce round to perform a part of the computation and stops when a termination condition is met. This will create α pauses in a given computation for checking the termination condition and will produce accurate results without requiring redundant work done on each processing node.

- **Performance Estimation:** Setting α and γ to some small value will incur small parallelism overhead and may cause effective resource utilization by allowing to perform large number of computations in parallel. But this may also require a large amount of memory on each mapper to hold map output. Setting these controls to some large value will reduce such memory requirements but will also incur large parallelism overhead which may offset the performance gain of parallel processing. Our analytical study helps estimating the cost of MR-RBAT for these parameter settings. Our performance estimation captures the characteristics of a physical architecture and given problem instance. Our performance estimation has practical implications. Our experimental study (Section 5.3) confirms that the effect of α and γ on parallelism overhead and computational cost is the same as predicted by our analytical study.

6.2 Future Work

This work has shown that our proposed methods can achieve scalability w.r.t large data volumes and the available resources. Our work can be extended in various directions:

- **Multiple Specializations:** A possible extension to MR-RBAT is to allow multiple specializations in parallel. Currently our methods preserves the sequence of key steps of RBAT. That is, starting with the most generalized item $\tilde{i}$, it splits a generalized item into two more specialized items $\tilde{i}_l$ and $\tilde{i}_r$ and performs the checking of all PS-rules. Next, the algorithm considers $\tilde{i}_l$ and $\tilde{i}_r$ for further specializations using the split-check iterations in sequence. To make more effective resource utilization, the specialization of $\tilde{i}_l$ and $\tilde{i}_r$ can be made in parallel.

- **Data utility estimation:** Our analytical study helps estimate the performance of MR-RBAT for given parameter settings but does not give an indication of how these parameter settings affect the utility loss of the anonymized data. Some applications such as biomedical studies [42] require high data utility. Therefore, choosing the parameter settings that make a good trade off between performance gain and the utility of anonymized data is important. Our method can be extended in this direction, to give some approximation guarantees for certain parameter settings.

- **Data Heterogeneity:** Our partitioning method assumes that the dataset to be anonymized only consists of the records containing a set of items only. For real-world applications, the datasets to be published contain both relational and transaction attributes [98]. For example, some analytical studies may require purchasing patterns from customers of certain age groups or gender. Unlike transactional data, relational data can contain numerical attributes such as age or average income. Dealing with such heterogenous data containing relational attributes and set of items requires a different partitioning strategy.

References

[1] A special report on managing information: Data, data everywhere. *The Economist*, February 2010.

[2] C.C. Aggarwal and P.S. Yu. *Privacy-Preserving Data Mining: Models and Algorithms*. Advances in Database Systems. Springer US, 2008.

[3] Rajendra Akerkar. *Big data computing*. CRC Press, 2013.

[4] Anita L. Allen. *Uneasy access : privacy for women in a free society / Anita L. Allen*. Rowman & Littlefield, Totowa, N.J. :, 1988.

[5] Michael Barbaro, Tom Zeller, and Saul Hansell. A face is exposed for aol searcher no. 4417749. *New York Times*, 9(2008):8For, 2006.

[6] James Bennett and Stan Lanning. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35, 2007.

[7] Erik-Oliver Blass, Roberto Di Pietro, Refik Molva, and Melek Önen. Prism–privacy-preserving search in mapreduce. In *Privacy Enhancing Technologies*, pages 180–200. Springer, 2012.

[8] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks*, 56(18):3825–3833, 2012.

[9] Yingyi Bu, Bill Howe, Magdalena Balazinska, and Michael D. Ernst. Haloop: Efficient iterative data processing on large clusters. *Proc. VLDB Endow.*, 3(1-2):285–296, September 2010.

[10] Ji-Won Byun and Ninghui Li. Purpose based access control for privacy protection in relational database systems. *The VLDB Journal*, 17(4):603–619, 2008.

[11] Duncan KG Campbell. A survey of models of parallel computation. *REPORT-UNIVERSITY OF YORK DEPARTMENT OF COMPUTER SCIENCE YCS*, 1997.

[12] Jianneng Cao, Panagiotis Karras, Chedy Raïssi, and Kian-Lee Tan. ρ-uncertainty: inference-proof transaction anonymization. *Proceedings of the VLDB Endowment*, 3(1-2):1033–1044, 2010.

[13] DM Carlisle, ML Rodrian, and CL Diamond. California inpatient data reporting manual, medical information reporting for california. Technical report, Tech. rep., Office of Statewide Health Planning and Development, 2007.

[14] Shuchi Chawla, Cynthia Dwork, Frank McSherry, Adam Smith, and Hoeteck Wee. Toward privacy in public databases. In *Theory of Cryptography*, pages 363–385. Springer, 2005.

[15] Hsinchun Chen, Roger HL Chiang, and Veda C Storey. Business intelligence and analytics: From big data to big impact. *MIS quarterly*, 36(4):1165–1188, 2012.

[16] Linchuan Chen and Gagan Agrawal. Optimizing mapreduce for gpus with effective shared memory usage. In *Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '12, pages 199–210, New York, NY, USA, 2012. ACM.

[17] Linchuan Chen, Xin Huo, and Gagan Agrawal. Accelerating mapreduce on a coupled cpu-gpu architecture. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 25. IEEE Computer Society Press, 2012.

[18] Rui Chen, Noman Mohammed, Benjamin CM Fung, Bipin C Desai, and Li Xiong. Publishing set-valued data via differential privacy. *Proceedings of the VLDB Endowment*, 4(11):1087–1098, 2011.

[19] Forrester Research. Privacy concerns cost ecommerce $15 billion(September). http://www.forrester.com/. 2001.

[20] David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Erik Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten Von Eicken.

LogP: Towards a realistic model of parallel computation, volume 28. ACM, 1993.

[21] Tore Dalenius. Finding a needle in a haystack or identifying anonymous census records. *Journal of official statistics*, 2(3):329, 1986.

[22] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[23] Christos Doulkeridis and Kjetil Nørvåg. A survey of large-scale analytical query processing in mapreduce. *The VLDB Journal-The International Journal on Very Large Data Bases*, 23(3):355–380, 2014.

[24] Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. John Wiley & Sons, 2012.

[25] Cynthia Dwork. Differential privacy: A survey of results. In *Theory and applications of models of computation*, pages 1–19. Springer, 2008.

[26] Cynthia Dwork. Differential privacy. In *Encyclopedia of Cryptography and Security*, pages 338–340. Springer, 2011.

[27] Jaliya Ekanayake, Hui Li, Bingjing Zhang, Thilina Gunarathne, Seung-Hee Bae, Judy Qiu, and Geoffrey Fox. Twister: a runtime for iterative mapreduce. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 810–818. ACM, 2010.

[28] Stanford encyclopedia of philosophy. http://plato.stanford.edu/entries/privacy/.

[29] Alina Ene, Sungjin Im, and Benjamin Moseley. Fast clustering using mapreduce. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 681–689. ACM, 2011.

[30] Jon Feldman, S Muthukrishnan, Anastasios Sidiropoulos, Cliff Stein, and Zoya Svitkina. On distributing symmetric streaming computations. *ACM Transactions on Algorithms (TALG)*, 6(4):66, 2010.

[31] Robson Leonardo Ferreira Cordeiro, Caetano Traina, Junior, Agma Juci Machado Traina, Julio López, U. Kang, and Christos Faloutsos. Clustering very large multi-dimensional datasets with mapreduce. In *Proceedings of the 17th*

ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '11, pages 690–698, New York, NY, USA, 2011. ACM.

[32] Michael J Flynn. Some computer organizations and their effectiveness. *Computers, IEEE Transactions on*, 100(9):948–960, 1972.

[33] Steven Fortune and James Wyllie. Parallelism in random access machines. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 114–118. ACM, 1978.

[34] Benjamin Fung, Ke Wang, Rui Chen, and Philip S Yu. Privacy-preserving data publishing: A survey of recent developments. *ACM Computing Surveys (CSUR)*, 42(4):14, 2010.

[35] Benjamin Fung, Ke Wang, and Philip S Yu. Top-down specialization for information and privacy preservation. In *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, pages 205–216. IEEE, 2005.

[36] Benjamin C. M. Fung, Ke Wang, Rui Chen, and Philip S. Yu. Privacy-preserving data publishing: A survey of recent developments. *ACM Comput. Surv.*, 42(4):14:1–14:53, June 2010.

[37] Johannes Gehrke, Raghu Ramakrishnan, and Venkatesh Ganti. Rainforest-a framework for fast decision tree construction of large datasets. In *VLDB*, volume 98, pages 416–427, 1998.

[38] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *ACM SIGOPS operating systems review*, volume 37, pages 29–43. ACM, 2003.

[39] Gabriel Ghinita, Panos Kalnis, and Yufei Tao. Anonymous publication of sensitive transactional data. *Knowledge and Data Engineering, IEEE Transactions on*, 23(2):161–174, 2011.

[40] Gabriel Ghinita, Yufei Tao, and Panos Kalnis. On the anonymization of sparse high-dimensional data. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pages 715–724. IEEE, 2008.

[41] Phillip B Gibbons. A more practical pram model. In *Proceedings of the first annual ACM symposium on Parallel algorithms and architectures*, pages 158–168. ACM, 1989.

[42] Aris Gkoulalas-Divanis and Grigorios Loukides. Medical data privacy handbook.

[43] David E Golberg. Genetic algorithms in search, optimization, and machine learning. *Addion wesley*, 1989, 1989.

[44] Michael T Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, searching, and simulation in the mapreduce framework. In *Algorithms and Computation*, pages 374–383. Springer, 2011.

[45] Mark Goudreau, Kevin Lang, Satish Rao, Torsten Suel, and Thanasis Tsantilas. Towards efficiency and portability: Programming with the bsp model. In *Proceedings of the eighth annual ACM symposium on Parallel algorithms and architectures*, pages 1–12. ACM, 1996.

[46] Ananth Y Grama, Anshul Gupta, and Vipin Kumar. Isoefficiency: Measuring the scalability of parallel algorithms and architectures. *IEEE concurrency*, (3):12–21, 1993.

[47] Patricia P Griffiths and Bradford W Wade. An authorization mechanism for a relational database system. *ACM Transactions on Database Systems (TODS)*, 1(3):242–255, 1976.

[48] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *ACM SIGMOD Record*, volume 29, pages 1–12. ACM, 2000.

[49] Bingsheng He, Wenbin Fang, Qiong Luo, Naga K Govindaraju, and Tuyong Wang. Mars: a mapreduce framework on graphics processors. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 260–269. ACM, 2008.

[50] Yeye He and Jeffrey F. Naughton. Anonymization of set-valued data via top-down, local generalization. *Proc. VLDB Endow.*, 2(1):934–945, August 2009.

[51] Yeye He and Jeffrey F Naughton. Anonymization of set-valued data via top-down, local generalization. *Proceedings of the VLDB Endowment*, 2(1):934–945, 2009.

[52] Dorit S Hochbaum and Anu Pathria. Analysis of the greedy approach in problems of maximum k-coverage. *Naval Research Logistics*, 45(6):615–627, 1998.

[53] Ihab F Ilyas, George Beskales, and Mohamed A Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys (CSUR)*, 40(4):11, 2008.

[54] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 59–72. ACM, 2007.

[55] Tochukwu Iwuchukwu and Jeffrey F. Naughton. K-anonymization as spatial indexing: Toward scalable and incremental anonymization. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB '07, pages 746–757. VLDB Endowment, 2007.

[56] Vijay S Iyengar. Transforming data to satisfy privacy constraints. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 279–288. ACM, 2002.

[57] Vijay S. Iyengar. Transforming data to satisfy privacy constraints. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 279–288, New York, NY, USA, 2002. ACM.

[58] Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 938–948. Society for Industrial and Applied Mathematics, 2010.

[59] Richard M. Karp. A survey of parallel algorithms for shared-memory machines. Technical report, Berkeley, CA, USA, 1988.

[60] George Karypis and Vipin Kumar. Multilevelk-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed computing*, 48(1):96–129, 1998.

[61] Ponnurangam Kumaraguru and Lorrie F. Cranor. Privacy Indexes: A Survey of Westin's Studies.

[62] Kyong-Ha Lee, Yoon-Joon Lee, Hyunsik Choi, Yon Dohn Chung, and Bongki Moon. Parallel data processing with mapreduce: A survey. *SIGMOD Rec.*, 40(4):11–20, January 2012.

[63] Kristen LeFevre and David DeWitt. Scalable anonymization algorithms for large data sets. *Age*, 40:40, 2007.

[64] Kristen LeFevre, David J DeWitt, and Raghu Ramakrishnan. Mondrian multidimensional k-anonymity. In *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*, pages 25–25. IEEE, 2006.

[65] Kristen LeFevre, David J. DeWitt, and Raghu Ramakrishnan. Workload-aware anonymization techniques for large-scale datasets. *ACM Trans. Database Syst.*, 33(3):17:1–17:47, September 2008.

[66] F Thomson Leighton. *Introduction to parallel algorithms and architectures: Arrays· trees· hypercubes*. Elsevier, 2014.

[67] Haoyuan Li, Yi Wang, Dong Zhang, Ming Zhang, and Edward Y Chang. Pfp: parallel fp-growth for query recommendation. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 107–114. ACM, 2008.

[68] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 106–115. IEEE, 2007.

[69] Tiancheng Li and Ninghui Li. Optimal k-anonymity with flexible generalization schemes through bottom-up searching. In *Data Mining Workshops, 2006. ICDM Workshops 2006. Sixth IEEE International Conference on*, pages 518–523. IEEE, 2006.

[70] Junqiang Liu and Ke Wang. Anonymizing transaction data by integrating suppression and generalization. In *Advances in Knowledge Discovery and Data Mining*, pages 171–180. Springer, 2010.

[71] Stuart P Lloyd. Least squares quantization in pcm. *Information Theory, IEEE Transactions on*, 28(2):129–137, 1982.

[72] Grigorios Loukides, Aris Gkoulalas-Divanis, and Bradley Malin. Anonymization of electronic medical records for validating genome-wide association studies. *Proceedings of the National Academy of Sciences*, 107(17):7898–7903, 2010.

[73] Grigorios Loukides, Aris Gkoulalas-Divanis, and Bradley Malin. Coat: Constraint-based anonymization of transactions. *Knowledge and Information Systems*, 28(2):251–282, 2011.

[74] Grigorios Loukides, Aris Gkoulalas-Divanis, and Jianhua Shao. Anonymizing transaction data to eliminate sensitive inferences. In *Database and Expert Systems Applications*, pages 400–415. Springer, 2010.

[75] Grigorios Loukides, Aris Gkoulalas-Divanis, and Jianhua Shao. Efficient and flexible anonymization of transaction data. *Knowledge and information systems*, 36(1):153–210, 2013.

[76] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. L-diversity: Privacy beyond k-anonymity. *ACM Trans. Knowl. Discov. Data*, 1(1), March 2007.

[77] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA., 1967.

[78] Travis Mayberry, Erik-Oliver Blass, and Agnes Hui Chan. Pirmap: Efficient private information retrieval for mapreduce. In *Financial Cryptography and Data Security*, pages 371–385. Springer, 2013.

[79] Andrew W McNabb, Christopher K Monson, and Kevin D Seppi. Parallel pso using mapreduce. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 7–14. IEEE, 2007.

[80] O.R.O.R. Media. *Big Data Now: 2012 Edition*. CreateSpace Independent Publishing Platform, 2014.

[81] Neelam Memon, Grigorios Loukides, and Jianhua Shao. A parallel method for scalable anonymization of transaction data. In *2015 14th International Symposium on Parallel and Distributed Computing*, pages 235–241. IEEE, 2015.

[82] Neelam Memon and Jianhua Shao. Mr-rbat: Anonymizing large transaction datasets using mapreduce. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 3–18. Springer, 2015.

[83] Kasper Mullesgaard, Jens Laurits Pedersen, Hua Lu, and Yongluan Zhou. Efficient skyline computation in mapreduce. In *17th International Conference on Extending Database Technology (EDBT)*, pages 37–48, 2014.

[84] Arvind Narayanan and Vitaly Shmatikov. How to break anonymity of the netflix prize dataset. *CoRR*, abs/cs/0610105, 2006.

[85] Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large sparse datasets. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pages 111–125. IEEE, 2008.

[86] Mehmet Ercan Nergiz, Maurizio Atzori, and Chris Clifton. Hiding the presence of individuals from shared databases. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 665–676. ACM, 2007.

[87] Mehmet Ercan Nergiz, Christopher Clifton, and Ahmet Erhan Nergiz. Multirelational k-anonymity. *Knowledge and Data Engineering, IEEE Transactions on*, 21(8):1104–1117, 2009.

[88] House of Commons Health Commity. The Electronic Patient Record. http://www.publications.parliament.uk/pa/cm200607/cmselect/cmhealth/422/422.pdf. 2007.

[89] Alper Okcan and Mirek Riedewald. Processing theta-joins using mapreduce. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 949–960. ACM, 2011.

[90] Canada CKAN open government data from US, EU and more. http://dataportals.org/.

[91] Canada CKAN Open Government data from US, EU and more. http://dataportals.org/.

[92] Paolo Palmerini, Salvatore Orlando, and Raffaele Perego. Statistical properties of transactional databases. In *Proceedings of the 2004 ACM symposium on Applied computing*, pages 515–519. ACM, 2004.

[93] Biswanath Panda, Joshua S Herbach, Sugato Basu, and Roberto J Bayardo. Planet: massively parallel learning of tree ensembles with mapreduce. *Proceedings of the VLDB Endowment*, 2(2):1426–1437, 2009.

[94] Spiros Papadimitriou and Jimeng Sun. Disco: Distributed co-clustering with map-reduce: A case study towards petabyte-scale end-to-end mining. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 512–521. IEEE, 2008.

[95] W. A. Parent. A new definition of privacy for the law. *Law and Philosophy*, 2(3):305–338, 1983.

[96] Yoonjae Park, Jun-Ki Min, and Kyuseok Shim. Parallel computation of skyline and reverse skyline queries using mapreduce. *Proceedings of the VLDB Endowment*, 6(14):2002–2013, 2013.

[97] Greg Pass, Abdur Chowdhury, and Cayley Torgeson. A picture of search. In *Proceedings of the 1st International Conference on Scalable Information Systems*, InfoScale '06, New York, NY, USA, 2006. ACM.

[98] Giorgos Poulis, Grigorios Loukides, Aris Gkoulalas-Divanis, and Spiros Skiadopoulos. Anonymizing data with relational and transaction attributes. In *Machine Learning and Knowledge Discovery in Databases*, pages 353–369. Springer, 2013.

[99] Matteo Riondato, Justin A DeBrabant, Rodrigo Fonseca, and Eli Upfal. Parma: a parallel randomized algorithm for approximate association rules mining in mapreduce. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 85–94. ACM, 2012.

[100] Shariq J. Rizvi and Jayant R. Haritsa. Maintaining data privacy in association rule mining. In *Proceedings of the 28th International Conference on Very Large Data Bases*, VLDB '02, pages 682–693. VLDB Endowment, 2002.

[101] Kenneth Rose, Eitan Gurewitz, and Geoffrey Fox. A deterministic annealing approach to clustering. *Pattern Recognition Letters*, 11(9):589–594, 1990.

[102] Indrajit Roy, Srinath T. V. Setty, Ann Kilzer, Vitaly Shmatikov, and Emmett Witchel. Airavat: Security and privacy for mapreduce. In *Proceedings of the*

7th USENIX Conference on Networked Systems Design and Implementation, NSDI'10, pages 20–20, Berkeley, CA, USA, 2010. USENIX Association.

[103] Tesco sells details of your shopping habits for Â£53m. http://www.dailymail.co.uk/news/article-1365512/tesco-sells-details-shopping-habits-53m.html. 2015.

[104] David B Skillicorn. Deriving parallel programs from specifications using cost information. *Science of Computer Programming*, 20(3):205–221, 1993.

[105] David B Skillicorn, Jonathan Hill, and William F McColl. Questions and answers about bsp. *Scientific Programming*, 6(3):249–274, 1997.

[106] DB Skillicorn. *Practical parallel computation*. Queen's University of Kingston. Department of Computing and Information Science, 1991.

[107] Rhys Smith and Jianhua Shao. Privacy and e-commerce: a consumer-centric perspective. 7(2):89–116, June 2007.

[108] Jeff Stuart, John D Owens, et al. Multi-gpu mapreduce on gpu clusters. In *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pages 1068–1079. IEEE, 2011.

[109] Latanya Sweeney. Achieving k-anonymity privacy protection using generalization and suppression. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):571–588, 2002.

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

[111] David Taniar, Clement HC Leung, Wenny Rahayu, and Sushant Goel. *High performance parallel database processing and grid databases*, volume 67. John Wiley & Sons, 2008.

[112] Khaled Tannir. *Optimizing Hadoop for MapReduce*. Packt Publishing Ltd, 2014.

[113] Manolis Terrovitis, Nikos Mamoulis, and Panos Kalnis. Privacy-preserving anonymization of set-valued data. *Proc. VLDB Endow.*, 1(1):115–125, August 2008.

[114] Manolis Terrovitis, Nikos Mamoulis, and Panos Kalnis. Privacy-preserving anonymization of set-valued data. *Proceedings of the VLDB Endowment*, 1(1):115–125, 2008.

[115] Manolis Terrovitis, Nikos Mamoulis, and Panos Kalnis. Local and global recoding methods for anonymizing set-valued data. *The VLDB Journal*, 20(1):83–106, February 2011.

[116] Manolis Terrovitis, Nikos Mamoulis, John Liagouris, and Spiros Skiadopoulos. Privacy preservation by disassociation. *Proc. VLDB Endow.*, 5(10):944–955, June 2012.

[117] Manolis Terrovitis, Nikos Mamoulis, John Liagouris, and Spiros Skiadopoulos. Privacy preservation by disassociation. *Proceedings of the VLDB Endowment*, 5(10):944–955, 2012.

[118] Wade Trappe and Lawrence C Washington. *Introduction to cryptography with coding theory*. Pearson Education India, 2006.

[119] Godwin J Udo. Privacy and security concerns as major barriers for e-commerce: a survey study. *Information Management & Computer Security*, 9(4):165–174, 2001.

[120] Opening up Goverments: publicly available data from United Kingdom. https://data.gov.uk/.

[121] Leslie G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, August 1990.

[122] Abhishek Verma, Xavier Llora, David E Goldberg, and Roy H Campbell. Scaling genetic algorithms using mapreduce. In *Intelligent Systems Design and Applications, 2009. ISDA'09. Ninth International Conference on*, pages 13–18. IEEE, 2009.

[123] Rares Vernica, Michael J. Carey, and Chen Li. Efficient parallel set-similarity joins using mapreduce. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 495–506, New York, NY, USA, 2010. ACM.

[124] Ke Wang, Benjamin C. M. Fung, and Philip S. Yu. Template-based privacy preservation in classification problems. In *Proceedings of the Fifth IEEE International Conference on Data Mining*, ICDM '05, pages 466–473, Washington, DC, USA, 2005. IEEE Computer Society.

[125] Ke Wang, Benjamin C. M. Fung, and Philip S. Yu. Handicapping attacker's confidence: An alternative to k-anonymization. *Knowl. Inf. Syst.*, 11(3):345–368, April 2007.

[126] Ke Wang, Philip S Yu, and Sourav Chakraborty. Bottom-up generalization: A data mining solution to privacy protection. In *Data Mining, 2004. ICDM'04. Fourth IEEE International Conference on*, pages 249–256. IEEE, 2004.

[127] Ting Wang, Shicong Meng, Bhuvan Bamba, Ling Liu, and Calton Pu. A general proximity privacy principle. In *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*, pages 1279–1282. IEEE, 2009.

[128] Alan F. Westin. *Privacy and freedom / [by] Alan F. Westin ; foreword by Oscar M. Ruebhausen.* Atheneum, New York :, [1st ed.] edition, 1967.

[129] Raymond Chi-Wing Wong, Jiuyong Li, Ada Wai-Chee Fu, and Ke Wang. (&#945;, k)-anonymity: An enhanced k-anonymity model for privacy preserving data publishing. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 754–759, New York, NY, USA, 2006. ACM.

[130] C Xavier and Sundararaja S Iyengar. *Introduction to parallel algorithms*, volume 1. John Wiley & Sons, 1998.

[131] Jian Xu, Wei Wang, Jian Pei, Xiaoyuan Wang, Baile Shi, and Ada Wai-Chee Fu. Utility-based anonymization using local recoding. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 785–790, New York, NY, USA, 2006. ACM.

[132] Yabo Xu, Ke Wang, Ada Wai-Chee Fu, and Philip S. Yu. Anonymizing transaction databases for publication. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, pages 767–775, New York, NY, USA, 2008. ACM.

[133] Hung-chih Yang, Ali Dasdan, Ruey-Lung Hsiao, and D Stott Parker. Map-reduce-merge: simplified relational data processing on large clusters. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1029–1040. ACM, 2007.

[134] Andrew Yao. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 162–167. IEEE, 1986.

[135] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, volume 10, page 10, 2010.

[136] Kehuan Zhang, Xiaoyong Zhou, Yangyi Chen, XiaoFeng Wang, and Yaoping Ruan. Sedic: privacy-aware data intensive computing on hybrid clouds. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 515–526. ACM, 2011.

[137] Xiaobing Zhang, Wanchun Dou, Jian Pei, Surya Nepal, Chao Yang, Cong Liu, and Jiann-Jong Chen. Proximity-aware local-recoding anonymization with mapreduce for scalable big data privacy preservation in cloud. 2013.

[138] Xuyun Zhang, Chang Liu, S. Nepal, Chi Yang, Wanchun Dou, and Jinjun Chen. Combining top-down and bottom-up: Scalable sub-tree anonymization over big data using mapreduce on cloud. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on*, pages 501–508, July 2013.

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