Top Down Specialization on Apache Spark™

Macarious Abadeer
School of Computer Science
Carleton University
Ottawa, Canada K1S 5B6
macariousabadeer@cmail.carleton.ca

November 30, 2019

Abstract

A very concise summary of the problem addressed and solution presented in this paper.

1 Introduction

Since the introduction of multi-core processors in 2004 by Intel®, parallel computing evolved to exploit the advantages of multiple processing units that became the norm for personal computers. This evolution was also expanded and accelerated by the advancements in Cloud Computing that supported running compute-intensive applications over a network of clusters. Parallel computing enabled the development of solutions to different real world applications that were hindered by scalability limitations such as big data analytics, machine learning and artificial intelligence. One of the problems that parallel computing provided scaleable solutions for is data anonymization, especially for big data.

In today's abundance of big data ranging from retail and banking transactions, health care, social media interactions and sensor data, a need was created for measures that protect people's most private and sensitive data. One of the most popular theories that were developed in this area was k-anonymity developed by Samarati and Sweeney in 1998 [14]. Sweeney argued that an individual in a dataset can be identified when the dataset is linked with other public datasets even if the original dataset did not contain identifying information such as name, date of birth and social insurance number. Sweeney was able to show that when linking voter registration cards and health care data, individuals can be identified with 87% accuracy. Those potentially identifying attributes are called Quasi-Identifiers (QID). k-anonymity states that a dataset is called k-anonymous when for a given record, there exists at least $k-1$ records in the same dataset with the same QID values. Further modifications to k-anonymity were made to overcome its shortcomings such as introducing ℓ-diversity [8] and t-closeness [7]. ℓ-diversity ensures that sensitive attributes, such as diagnosis in a health care dataset, need to have diverse values so that an adversary with foreknowledge of a given QID set cannot deduce their diagnosis. t-closeness ensures that the distribution of these diverse values is close to their distribution in the original dataset.

While these theories contributed immensely to the practices of data anonymization, k-anonymity was proven to be $\mathcal{NP}$-hard by Meyerson and Williams [10]. Further research

used these models as a baseline to develop scalable parallel algorithms that can handle big data.

The paper is organized as follows: in Section 2, I will go over the different ideas that were proposed to optimize and scale k-anonymity. Section 4 will detail an implemented proposed solution and Section 5 presents the experimentation results of the algorithm and the paper finally concludes in Section 6.

2 Literature Review

There are three different masking types that are used to satisfy k-anonymity: interval, taxonomy tree and suppression [1]. Suppression requires certain outlier tuples to be removed to satisfy k-anonymity [14]. Intervals and taxonomy trees are generalization techniques applied to numerical and categorical attributes respectively [14]. For example two records with birth year of 1971 and 1973 can be generalized to 1970-1975. For a taxonomy tree, a categorical attribute such as education level can have, for example, post-graduate as a parent node which can have PhD, Masters and Post Graduate Diploma as its child leaves so that records with these values can be generalized to the parent node. The majority of research papers on anonymization with respect to big data involved taxonomy trees thus this is where I focus my literature survey.

One of the techniques that researchers attempted to optimize was Bottom-Up Generalization (BUG) which involves traversing the taxonomy tree of attribute hierarchies from the bottom (most specific) upwards (most general) [6]. Wang suggested that the taxonomy tree would be provided by the data supplier or the data recipient [6]. As the tree is traversed, two metrics are calculated to ensure a high quality generalization: information loss and anonymity gain. An indexed approach to bottom-up generalization was proposed by Hoang [5] where the taxonomy tree was generated automatically at runtime. Hoang's indexed approach could also handle numerical as well as categorical attributes. Indexed BUG starts with collecting statistical information about the dataset as well as partition it so that it can be used in the generalization step which was further broken down to four steps: calculate the best generalization score based on the least information loss, calculate k-anonymity for every partition, generate an indexed generalization map which maps every value to its generalized value, and the last step creates the anonymized dataset using this map. Hoang's experiments showed that the generalization time did not increase with the dataset size due to the use of indexed generalization map however performance was impacted by the distinct values count for each QID [5].

Parallel BUG was introduced to address the limitations of traditional and indexed BUG approaches. Pandilakshmi attempted to solve the limitations of indexing structures since they are centralized and hard to parallelize and cannot run on distributed systems such as the Cloud [11]. Pandilakshmi introduced Bi-Level BUG algorithm where MapReduce framework was used to take advantage of job-level and task-level parallelization. Job-level parallelization was achieved by using multiple MapReduce jobs and task-level parallelization was achieved by using multiple mapper/reducer tasks for every MapReduce job so that they are executed in parallel on every partition. Data is partitioned according to a random number generated between 1 and p where p is the number of partitions. Pandilakshmi then runs MapReduce BUG driver (MRBUG) iteratively on the partitioned datasets and calculates generalization score (least information loss with the most anonymity gain) and stops until it finds the best generalization with the highest score that satisfies k-anonymity. Pandilak-

shmi experiments performed on varying datasets of up to 4GB showed that execution time was virtually capped at ≈33 minutes regardless of dataset size.

Another technique is Top-Down Specialization (TDS). TDS traverses the taxonomy tree from the top downwards where it starts with the most generalized values and specializes the value and stops when it violates k-anonymity [4]. Multiple solutions have been developed such as a scalable two-phase TDS introduced by [13] and [18]. The first phase involves partitioning the original dataset to p partitions using random sampling. A MapReduce TDS job runs in parallel on each partition. Each job specializes the data iteratively while calculating information gain and privacy loss metrics and creates an intermediate anonymized dataset. In the second phase the intermediate datasets are merged and further anonymized if necessary to satisfy k-anonymity. In [18], Zhang et al. adopted Hadoop® and took advantage of distributed cache capability to pass the intermediate anonymized dataset to each mapper/reducer node. The experiments for this solution showed an overhead in the partitioning phase of the dataset.

A hybrid approach of BUG and TDS using MapReduce was introduced by [17] where it was shown that when either TDS or BUG were used individually, they performed poorly for certain values of k. The hybrid approach applies TDS for large k values and BUG for smaller ones. The notion of Workload Balancing Point was introduced which is defined as the point where the amount of computation required for TDS is the same as BUG. Once that point is identified, the hybrid approach chooses TDS for k greater than the workload balancing point and chooses BUG when k is smaller. The workload balancing point is estimated using the height of the taxonomy tree as a reference.

Al-Zobbi et al [1] argued that finding the highest scoring generalization and specialization based on information gain and anonymity loss in BUG and TDS require high computational costs and impedes the ability to parallelize them. Al-Zobbi also argued that as the data grows in size, the high accuracy of these computations no longer make a statistical difference. Al-Zobbi proposed a multi-dimensional sensitivity-based algorithm on Apache Spark that uses a pre-determined QID attributes to anonymize as well as precalculated k value using linear regression. The solution also takes into consideration the probability value of each QID. For example assuming that age can range between 1 and 100, the probability of finding a given age is 1% which is much higher than a probability of finding a given job title assuming there are 200 different job titles. The solution prioritized the anonymization of higher probability attributes instead of calculating information gain and anonymity loss scores for every attribute. The solution also used a role-based access control equivalent system to set k based on context. For example a health care dataset maybe given a lower k (less anonymization) when shared with a doctor but a higher k value when shared with an insurance risk analyst. The solution was implemented on Spark and aimed at minimizing the use of User Defined Functions (UDFs) since they run outside of the Spark JVM which is beyond the resource negotiator's control. Al-Zobbi recognized that this solution would sacrifice the analytical value of the dataset for the performance improvement gained by not calculating the best generalization options.

In a research paper by [15], a survey was done on MapReduce vs. Spark for big data analytics. It concluded that Spark is better suited for problems that require accessing the same dataset multiple times such as the case with both TDS, BUG and their variants. The constant read and write by Hadoop to HDFS (Hadoop Distributed File System) is considered a significant overhead however Spark operates on datasets in memory and provides the capability of caching Resilient Distributed Datasets (RDDs) for faster access making it suitable for iterative algorithms. The experiments carried out by Shi et al in [15] found that

Spark is 5 times faster than MapReduce for iterative algorithms regardless of data size.

Shi's findings in [15] are inline with other researchers that implemented anonymization algorithms on Spark such as [2] and [16]. For example, [16] proposed a TDS implementation for Apache Spark that partitioned the dataset to p partitions on n Spark nodes where $n = p$. The master node partitions the data and calculates the scores required by TDS such as information gain and privacy loss. The scores are sent to the driver node which performs aggregations required by further iterations until k is satisfied. The experiments carried out for this solution by [16] showed that there is an overhead cost incurred when having more than one partition in a single node. The experiments also showed performance gains regardless of k values and dataset sizes as long as Spark nodes are increased with the dataset size. As outlined by [1], ideally the partitioned dataset needs to fit in the node's memory in order to avoid spilling to disk.

The previously mentioned solutions are generic enough to be applied to any type of datasets. However, multiple other solutions have been proposed to address specific anonymization scenarios. I briefly include them here due to their relevance in terms of parallelization techniques. Parameshwarappa [12] for example proposed a solution to anonymize physical activity collected by wearable gadgets. It uses a multi-level clustering algorithm based on Maximum Distance to Average Vector (MDAV). It attempts to cluster data points so that every cluster satisfies k-anonymity. If a cluster does not satisfy k-anonymity, differential privacy technique is used to add statistical noise to the cluster in a way that does not skew the analytical value of the dataset.

Another solution was implemented to provide a parallel anonymization of transaction data such as retail and banking datasets in [9]. It uses an algorithm known as RBAT on MapReduce which uses set-based generalization to anonymize data based on user-provided set of rules. It partitions data in a way that ensures the workload of every partition is approximately the same across different partitions. The solution scans the whole dataset in order to achieve this efficient partitioning based on QID values to minimize data shuffling across partitions.

Other frameworks were also developed to address specific variations to k-anonymity mentioned such as t-closeness introduced by [7]. For example, [3] developed a framework called Incognito using MapReduce that generates a distribution of sensitive attributes based on their count in the dataset. Given the frequency histogram generated, subsets of the sensitive attribute values that have the same parent in the taxonomy tree are put together in the same data bucket. The tree is sorted from left to right nodes in an ascending order of their frequencies in the generated histogram. The anonymized dataset is then mapped to the generated tree in order to ensure anonymized dataset is close to the original dataset in terms of distribution of values.

3 Problem Statement

This paper tackles the scalability of anonymization algorithms for Big Data specifically Top Down Specialization algorithm. There was only a handful of papers in the literature reviewed in Section 2 that implemented anonymization algorithms on Spark and only one that implemented Top Down Specialization [16]. In that implementation the performance was assessed only up to 500 MB of data which is relatively small in the context of Big Data. This paper aims to assess the performance of Top Down Specialization on Spark for datasets larger than 500 MB. It will also use number of records as the gauge instead of

size on disk. The question I aim to answer, how does Top Down Specialization scale up for datasets larger than 5 million rows or 500 MB? Are there any optimizations that can be done to improve speedups? Are there any new Spark features that were developed since the time when [16] was implemented that could improve performance?

4 Proposed Solution

This part of the paper is much more free-form. It may have one or several sections and subsections. But it all has only one purpose: to convince the reader that you answered the question or solved the problem that you set for yourself. In this section you will for example present new algorithms you developed and your implementation of these new algorithms.

4.1 Introduction to k-anonymity

It is important to review definitions that will be used throughout this paper.

Definition 1 (k-anonymity) *A dataset is called k-anonymous if for every record there exists at least $k-1$ other records with the same Quasi-Identifier values.*

Definition 2 (Quasi-Identifiers) *Quasi-Identifiers are attributes that do not directly identify an individual, but when used together and linked with other datasets they have the potential of identifying an individual. They will be referred to as QID throughout the paper.*

Definition 3 (Sensitive Attributes) *Sensitive Attributes are attributes that should remain private. They will be referred to as SA throughout the paper.*

Definition 4 (Taxonomy Trees) *Taxonomy Trees are logical hierarchies of distinct values in a dataset.*

For example, in Table 4.1 *Education*, *Gender* and *City* are examples of QIDs while *Income* is an example of SA. Table 4.1 does not satisfy k-anonymity since there are two unique records with the same QID values. For example, an adversary with foreknowledge of the existence of an Orleans Male with a Master's degree in the dataset will be able to deduce the individual's income. The two records violating k-anonymity are highlighted in Table 4.1.

Education	Gender	City	Income
Grade 12	Female	Nepean	$65,000
Bachelor's	Male	Ottawa	$50,000
Master's	Male	Orleans	$50,000
PhD	Male	Gloucester	$100,000
Grade 12	Female	Nepean	$80,000
Associate	Female	Kanata	$90,000
Associate	Female	Kanata	$105,000
Bachelor's	Male	Ottawa	$50,000

Table 1: Dataset violating k-anonymity

Figure 1 represents a taxonomy tree for *Education* column. The leaf nodes in the tree represent the distinct values present in the dataset. Taxonomy trees are provided by either the data provider or the data recipient for all QID in the dataset. The root node of all taxonomy trees is *Any*.

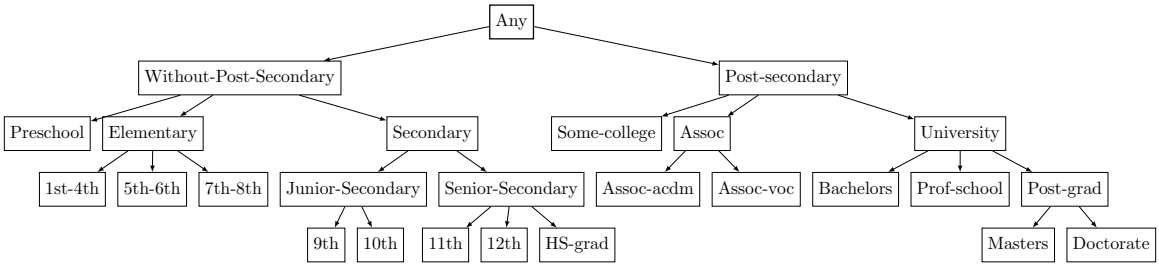

Figure 1: Education Taxonomy Tree

4.2 Top Down Specialization

Top down specialization algorithm is comprised of the following steps:

1. All non-QIDs are removed from dataset
2. QIDs and SAs are grouped together and count is calculated for every group
3. A set of Anonymization Levels AL is created. The set initially includes the taxonomy trees of all attributes.
4. All QID values are generalized to the root of the respective AL
5. A score is calculated for every AL using Equation 1
6. The AL with the highest score is selected, the root is removed from the AL set and its children are added
7. Iterate 4 through 6 until k is violated

$$Score(\nu) = \begin{cases} \frac{InfoGain(\nu)}{PrivacyLoss(\nu)} & PrivacyLoss(\nu) \neq 0 \\ InfoGain(\nu) & otherwise \end{cases} \tag{1}$$

$$InfoGain(\nu) = I(R_\nu) - \sum_{c \in children(\nu)} \frac{|R_{\nu c}|}{|R_\nu|} I(R_{\nu c}) \tag{2}$$

$$I(R_\nu) = - \sum_{s\nu \in \{SA\}} \frac{|R_{\nu s\nu}|}{|R_\nu|} \times \log_2 \frac{|R_{\nu s\nu}|}{|R_\nu|} \tag{3}$$

$$PrivacyLoss(\nu) = k(R_\nu) - k(R_{\nu c}) \tag{4}$$

4.3 Pre-Processing

Preprocessing, building path maps

4.4 Main Algorithm

Pseudo code, discussion

4.5 Performance enhancements

Performance enhancements, partitioning, having one aggregation per iteration

5 Experimental Evaluation

This section is not mandatory for all papers (for example theory papers) but typically required for papers in the field of parallel computing. After all, parallel computing is all about compute performance. Here you present performance data obtained from e.g. running your newly developed algorithms and code on a parallel machine using some benchmark input data. Typically, you need to describe the parallel machine you used and the data that you used as input. The main results are usually performance graphs, typically speedup curves. You want to evaluate your code on different input data sets highlighting the strengths and weaknesses of your code. Don't just use best case scenarios. People will call you on that. Discuss the results obtained.

6 Conclusions

You generally cover three things in the Conclusions section.

1. Conclusions
2. Summary of Contributions
3. Future Research

Conclusions are not a rambling summary of the thesis: they are short, concise statements of the inferences that you have made because of your work. All conclusions should be directly related to the research question.

The Summary of Contributions will be much sought and carefully read by the readers. Here you list the contributions of new knowledge that your paper makes. Of course, the paper itself must substantiate any claims made here. There is often some overlap with the Conclusions, but that's okay.

The Future Research should indicate interesting new problems arising from your work. No paper ever solves everything. In fact, the best research papers lead to new research questions for other researchers to work on.

References

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

[2] Y. Canbay and S. Sağıroğlu. Big data anonymization with spark. In *2017 International Conference on Computer Science and Engineering (UBMK)*, pages 833–838, October 2017.

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

[4] B. C. M. Fung, K. Wang, and P. S. Yu. Top-down specialization for information and privacy preservation. In *21st International Conference on Data Engineering (ICDE'05)*, pages 205–216, April 2005.

[5] A. Hoang, M. Tran, A. Duong, and I. Echizen. An indexed bottom-up approach for publishing anonymized data. In *2012 Eighth International Conference on Computational Intelligence and Security*, pages 641–645, November 2012.

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

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

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

[9] N. Memon, G. Loukides, and J. Shao. A parallel method for scalable anonymization of transaction data. In *2015 14th International Symposium on Parallel and Distributed Computing*, pages 235–241, June 2015.

[10] Adam Meyerson and Ryan Williams. On the complexity of optimal k-anonymity. In *Proceedings of the Twenty-third ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '04, pages 223–228, New York, NY, USA, 2004. ACM.

[11] K Pandilakshmi and G Rashitha Banu. An advanced bottom up generalization approach for big data on cloud. *Int J Comput Algor*, 3:1054–9, 2014.

[12] Pooja Parameshwarappa, Zhiyuan Chen, and Gunes Koru. A multi-level clustering approach for anonymizing large-scale physical activity data. *arXiv*, 2019.

[13] Zorige Priyanka, K Nagaraju, and Y Venkateswarlu. Data anonymization using map reduce on cloud based a scalable two-phase top-down specialization. *International Journal on Recent and Innovation Trends in Computing and Communication*, 2(12):3879–3883, 2014.

[14] Sweeney L. Samarati P. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and supression. Technical report, Massachusetts Institute of Technology and SRI International, 1998.

[15] Juwei Shi, Yunjie Qiu, Umar Farooq Minhas, Limei Jiao, Chen Wang, Berthold Reinwald, and Fatma Özcan. Clash of the titans: Mapreduce vs. spark for large scale data analytics. *Proc. VLDB Endow.*, 8(13):2110–2121, September 2015.

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

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

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