electronics

MDPI

Article

Scalable, High-Performance, and Generalized Subtree Data Anonymization Approach for Apache Spark

Sibghat Ullah Bazai, Julian Jang-Jaccard * and Hooman Alavizadeh

Cybersecurity Lab, Computer Science/Information Technology, Massey University, Auckland 0632, New Zealand; s.bazai@massey.ac.nz (S.U.B.); h.alavizadeh@massey.ac.nz (H.A.)
* Correspondence: j.jang-jaccard@massey.ac.nz

Abstract: Data anonymization strategies such as subtree generalization have been hailed as techniques that provide a more efficient generalization strategy compared to full-tree generalization counterparts. Many subtree-based generalizations strategies (e.g., top-down, bottom-up, and hybrid) have been implemented on the MapReduce platform to take advantage of scalability and parallelism. However, MapReduce inherent lack support for iteration intensive algorithm implementation such as subtree generalization. This paper proposes Distributed Dataset (RDD)-based implementation for a subtree-based data anonymization technique for Apache Spark to address the issues associated with MapReduce-based counterparts. We describe our RDDs-based approach that offers effective partition management, improved memory usage that uses cache for frequently referenced intermediate values, and enhanced iteration support. Our experimental results provide high performance compared to the existing state-of-the-art privacy preserving approaches and ensure data utility and privacy levels required for any competitive data anonymization techniques.

Keywords: Spark; subtree generalization; privacy; data anonymization; Resilient Distributed Dataset (RDD)

Citation: Bazai, S.U.; Jang-Jaccard, J.; Alavizadeh, H. Scalable, High-Performance and Generalized Subtree Data Anonymization Approach for Apache Spark. *Electronics* **2021**, *10*, 589. http://doi.org/10.3390/electronics10050589

Academic Editor: Jordi Guitart

Received: 5 December 2020
Accepted: 25 February 2021
Published: 3 March 2021

1. Introduction

Privacy preservation is an ongoing and challenging issue that impacts people's lives on a daily basis. This has inspired and motivated many computer science researchers to provide information privacy preservation approaches such as access restriction, encryption, noise induction, and data anonymization [1–3]. The access restriction approach only allows authorized entities to access data, while the encryption approach uses ciphers to protect data privacy. The noise induction approach modifies the original data with additional noise to protect privacy. However, Anonymization approaches such as *k*-Anonymization generalized or suppresses the sensitive information from the data to provide high utility and more privacy.

k-anonymization-based subtree generalization provides high data utility and better privacy strategies for single dimensional data when compared to full-tree generalization [4–6]. The iterative nature of subtree generalization is well suited to find a more efficient attribute generalization strategy. However, the complexity of execution time grows on each additional iteration increase for finding the optimal generalization level. The cost of computation will increase more when other aspects of anonymization are involved, for example, a *k*-group size, the number of attributes, and generalization hierarchy's tree.

Many solutions have been proposed for scalable big data anonymization [7–10]. Existing approaches of subtree data anonymization are mostly based on MapReduce platforms to take advantage of the scalability and cost-efficiency [11–13]. The MapReduce paradigm typically relies on the processing of two primary functions map and reduces where the former works as a sub-unit of data processing while the latter accumulates and produces the final data analytic results. Without appropriate support for algorithms that runs an extensive iteration such as subtree, the maps and reducers require to communicate many

times over, often sequentially and also fetching data from disk, which creates tremendous performance overheads [14,15].

An alternative approach, Spark [16] is used for addressing the overheads associated with MapReduce counterparts have been proposed, often comparing the performance results on both platforms [14,15,17,18]. In-memory-based Spark's performance has well been documented and proven effective for many iteration intensive algorithms such as seen in [19] where it demonstrated 10 times faster performance gain. Other approaches [14,15] also demonstrate the competitive performance advantage of Spark.

Close to our work, several proposals have emerged to illustrate the use of Spark for data anonymization techniques. For example, Ref [20] proposed a distributed Top-Down Specialization (TDS) algorithm that can work on Spark, and [21,22] proposed several sensitively-based multi-dimension data anonymization strategies to use Spark platform Sub tree anonymization. Anonylitics [23] used Spark's default iteration support to implement data anonymization and PRIMA [24] proposes a Spark anonymization strategy to define the utility and generalization level rules for limiting data loss. Although these existing proposals offer interesting aspects of the k-anonymity-based anonymization strategy, they neither provide any guidelines and strategies as to how different types of subtree data anonymization approaches can be best implemented using Spark as a generic framework nor provide any implications of privacy and utility measure.

In this paper, we propose a generic framework for implementing subtree-based data anonymization techniques on Apache Spark. The main contributions of this paper are as follows:

- We propose a Resilient Distributed Dataset (RDD)-based subtree generalization implementation strategy for Apache Spark. Our novel approach resolves the existing issues and can provide data anonymization outcomes regardless of any specific subtree implementation approaches (e.g., top-down, bottom-up, or hybrid);
- We clearly demonstrate how our proposal can reduce the complexity of operations and improve performance by the use of effective partition, improved memory and cache management for different types of intermediate values, and enhanced iteration support;
- We show that the proposed approach offers high scalability and performance through a better selection of subtree generalization process and data partitioning compared to the state-of-the-art similar approaches. We achieve high privacy and appropriate data utility by taking into account the data distribution and data processing using in-memory computation.
- Our intensive experiments results demonstrate the compatibility and application of our proposal on various datasets for privacy protection and high data utility. Our approach also outperforms the existing Spark-based approaches by providing the same privacy with minimum privacy loss.

The rest of this paper is organized as follows. Section 2 provides the related work and highlights the pros and cons of each similar work. Section 3 provides the background and definition used throughout the paper and discusses the details of the issues involved in existing subtree generalizations implemented in MapReduce. Section 4 describes the details of our proposal and clearly illustrates how our proposal can resolve the issues associated with the MapReduce-based approaches. In Section 5, we provide our experimental results including setup, configuration, and discuss the observations of the results. Finally, we conclude the paper in Section 6 and provide some potential future directions.

2. Related Work

Distributed anonymization methods are used to address the anonymization scalability. Most distributed algorithms presented so far aimed at meeting k-anonymity-based privacy models using distributed programming frameworks such as MapReduce. This motivated the authors of the present paper to develop a distributed method for satisfaction of privacy

and provide high data utility using subtree-based generalization that provides scalability and high-performance anonymization.

Subtree-based generalization can be broadly categorized into two kinds: Top-Down Specialization (TDS) [11] and Bottom-Up Generalization (BUG) [12]. In the TDS approach, the generalization typically starts from the topmost domain values in the taxonomy trees of attributes towards the bottom as an iterative process. In contrast, the techniques based on BUG generalize data from the bottom of the taxonomy tree towards its top, also iteratively. A hybrid approach that combines both TDS and BUG has been proposed [13]. The majority of these approaches so far have been implemented as sequential MapReduce jobs where the output of each MapReduce job is used as an input for subsequent steps until the anonymization constraints met. Such sequential execution of jobs can attribute significant performance overheads.

Several Spark-based approaches were proposed to address the concerns associated with MapReduce-based data anonymization strategies. Zaharia et al. [16] illustrated a competitive performance advantage of in-memory-based Spark operations compared to disk-based MapReduce execution. Their results demonstrated that Spark's implementation of iterative operations was 100 times faster than it was implemented under the MapReduce platform as Spark provides better parallelism by allowing many iterative tasks running at the same time often accessing memory instead of disks [14,15]. The authors in [14] provided benchmarking results using Word Count, *k*-means, and PageRank where Spark outperformed over MapReduce especially on iterative tasks. Their work stated that the performance gain of Spark was due to Resilient Distributed Dataset (RDD) caching that reduced the overheads associated with disk and CPU. Maillo et al. [15] demonstrated the performance advantage of Apache Spark on iterative tasks based on K-nearest Neighbour (KNN) using the datasets which contained 10 million instances.

Sopaoglu and Abul [20] developed a distributed TDS algorithm to provide *k*-anonymity that works for Apache Spark. The main focus of their study was to improve the scalability aspect of the original TDS algorithm [11] by offering improved partition management. Using the adult dataset, they evaluated that the scalability and run-time were significantly improved. Al-zobbi et al. [21,22] proposed several sensitivity-based multi-dimension anonymization strategies that could produce different levels of information obscurity depends on the different access privilege levels of the users (i.e., more customized data generalization result suitable for each user). To understand the roles and responsibility of the user accessing the system, the proposal used a User Defined Function (UDF) of Spark which allows the developer of Spark to be able to extend the vocabulary of default Spark SQL. Their proposal also illustrated that it was possible to reduce the data transmission time between memory and disks by serializing data with Spark RDD.

To address the overheads associated with MapReduce, several Spark-based approaches have been proposed in recent years [18,25–28]. In [29], the authors proposed the INCOGNITO framework for full-domain generalization using Spark RDDs. Although their experiential results illustrate the improvement in both scalability and execution efficiency, they did not provide any insights into privacy and utility trade-offs. Anonylitics [23] provides Spark's default iteration-based data anonymization implementation. The approach provides large-scale data anonymization; however, their approach does not address the potential memory exhaustion unable to accommodate an increasing number of intermediate data produced as the number of iterations increases. PRIMA [24] proposes a data anonymization strategy for Apache Spark with Optimal Lattice Anonymization (OLA). OLA provides data utility and generalization level rules in order to limit the data utility loss. However, the proposed approach does not provide performance comparison and privacy validation with existing approaches.

Somewhat similar but in a different realm of data anonymization technique using differential privacy [30] for Apache Spark, Gao et al. [26,31] proposed several techniques to anonymize k-means clustering algorithm on Spark platform. In their approaches, a new optimal partition mechanism is used to determine the dynamic allocation of datasets for

fast processing on the Spark platform. Different partitions containing different classes of datasets then are applied with noises based on Laplace calculation in the reduce phase. A formal privacy proof meeting ε-differential privacy requirement is described. Yin et al. [32] also proposed another approach for data anonymization that uses the Map-Reduce model to control the parallel distribution of k-means clustering and at the same time uses Laplace to implement differential privacy protection. In [33], the authors proposed a more holistic approach to produce differentially private datasets using a synthesizing program that can run on data-parallel analytics frameworks such as Apache Spark. Unlike these existing differential privacy-based approaches where the main focus of proposals is with providing a more solid theoretical foundation for privacy guarantee, our work focus on a mechanism to provide privacy protection of a published data.

3. Subtree Generalization

In this section, we describe the basic symbols and their descriptions used in this paper (see Table 1) together with the general algorithm involved in a subtree generalization.

Table 1. Symbols and Descriptions.

Symbol	Description	Reference
D	Dataset	Algorithm 1
D^*	Anonymized Dataset	Algorithm 1
r	Record	Algorithms 1–4
SA	Sensitive Attributes	Algorithm 1
RDD_in	Input_RDD	Algorithm 2
RDD^*	Anonymized_RDD	Algorithm 4
$\|D\|$	Total number of Record	Algorithm 4
n	nth Record	Algorithms 1–4
r^*	Anonymized Record	Algorithm 4
AL	Anonymization Level	Algorithms 3 and 4
C_r	Record count	Algorithms 2 and 3
C_r^*	Anonymized record count	Algorithm 4
A_v	Attribute Value	Section 4.2
TT	Taxonomy Tree	Algorithm 3
k	Anonymity Parameter	Algorithm 4
QID	Quasi-identifiers set	Algorithm 4
qid	Quasi-identifier	Section 4.2
C_A	Child attribute	Algorithm 3
P_A	Parent attribute	Algorithm 3
DOM	Domain value in TT	Algorithm 3

3.1. Preliminaries

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

3.2. Subtree Generalization Algorithm

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

As mentioned, Figure 1 represents the example of Taxonomy Trees (TT) based on Gender, Age, Job, and Education of the census dataset [34]. Each TT includes roots (parent nodes), middle nodes (in between the parent and child nodes but most often act the same as the parent nodes), and leaves (which are mostly child nodes). In a subtree scheme, generalizations are applied for the parent nodes if any child nodes are generalized. For example, in Figure 1b, if the 'Dancer' child node is generalized to its parent node 'Artist',

then another child node 'Writer' also needs to be generalized to 'Artist'. Please note that 'Engineer' and 'Lawyer' child nodes retain their values as the dimension of their parent node "Professional" is not affected. The root (parent) node of all taxonomy trees is often called 'Any'.

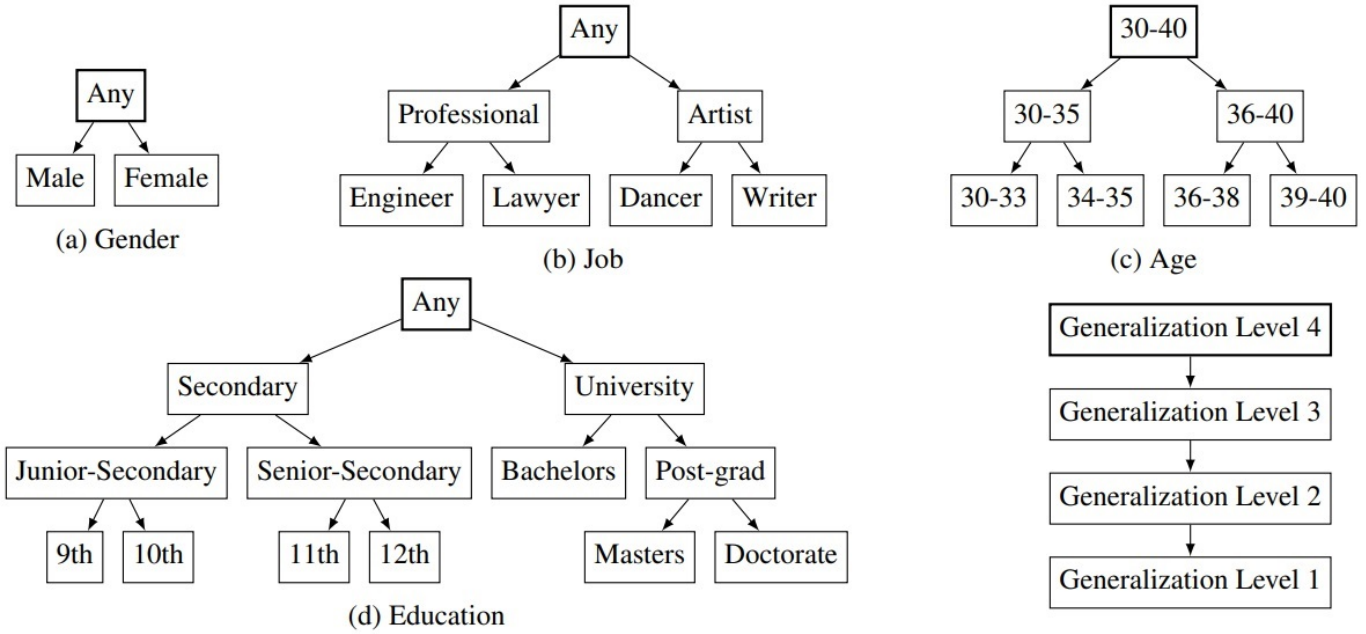

Figure 1. Examples of Taxonomy Trees.

Subtree generalizes data by applying one level of generalization at a time on an attribute by converting child node to parent node. The Subtree generalization steps are presented in Algorithm 1. The iteration starts from the child level. Then, at each step, a specific value (i.e., child) is generalized to a general value (i.e., parent) for an attribute within a QID. This process is repeated until the highest level of generalization violates k-anonymity rule [35]. Table 2 shows the original dataset along with the count of each record (i.e., the frequency of the same record appeared) in the database. Table 3 is produced from Table 2 as a result of a generalization level applied based on Taxonomy Trees depicted in Figure 1. After the first level of generalization, we observe that the attribute of Education for the child nodes "9th" and "10th" are generalized to "Junior-Secondary". Similarly, "Masters" and "Doctorate" child nodes are generalized to "Post-grad", and other child nodes remain the same in this round. Finally, this iteration process is repeated until all QID meet the final required anonymization level, as represented in Table 4.

Algorithm 1: Subtree Generalization Algorithm

Input: D
Output: D^*
begin
  QID, $SA \leftarrow r$ for i = 1 to n in D
  Compare k with count of r
  Compare Cut_i score for each QID_i select highest
  Replace QID_i child to QID_i parent in TT for Cut_i score
  Count the r with updated value
  Repeat step 3 to 6 until k is greater than the number of anonymized records
  return (D^*)
end

Table 2. A sample dataset.

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

Table 3. The result of the first generalization applied to the original dataset.

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

Table 4. Fully anonymized dataset applied to the original dataset.

Education	Gender	Age	Income	Count
Junior-Secondary	M	30–33	≤50 k	7
11th	M	35	>50 k	5
12th	F	37	>50 k	4
Bachelors	F	40–45	>50 k	10
Post-grad	Any	44	>5 0k	8

The overall subtree generalization algorithm is described in Algorithm 1. Each round of iteration includes four major steps: (i) Comparing the *k*-anonymity level with the number of records generalized, (ii) Calculating the data utility and privacy scores based on [6] for all *QIDs*, (iii) Finding the best generalization level by comparing the score values for all *QID*s and decide the next generalization level based on the highest score of a *QID*, and (iv) Applying the highest score of the *QID* and apply the generalization to all *QID*s in the same Equivalence Class.

3.3. Review of Subtree Implementation in MapReduce

In this section, we review the subtree implementation based on the MapReduce platform and extensively discuss the main limitations involved. There are four main phases in a typical subtree implementation that use MapReduce platform [12] (shown in Figure 1). MapReduce jobs contained in these four phases are coordinated together to accomplish the subtree anonymization. The description for each of the four main phases is as follows:

(1) *Partition MapReduce Job*: this phase involves dividing the original datasets into multiple chunks (i.e., partitions) in which each chunk contains a smaller portion of the original datasets.
(2) *MapReduce Job Intermediate Subtree*: This phase applies data anonymization to each chunk in parallel resulting in producing intermediate anonymized results.
(3) *MapReduce Job Combiner Subtree Result*: In this phase, MapReduce jobs combine all intermediate anonymized results to form an intermediate anonymized dataset.
(4) *MapReduce Job Final Subtree*: In this phase, the *k*-anonymity for the complete dataset is validated using the execution of two MapReduce jobs on the intermediate anonymized datasets.

The bold solid arrow lines represented in Figure 2 indicate how *parallelism* works in the MapReduce platform. It shows the timeline of the parallel processing for multiple MapReduce jobs which are executed in parallel on each node to process the intermediate data. The gray solid arrow lines show the MapReduce platform *execution flow* from mapping intermediate data to reducing phase while the dashed arrow lines describe the *data flow* from dispatching Anonymization Level into a cache for updating *k* at each iteration MapReduce round.

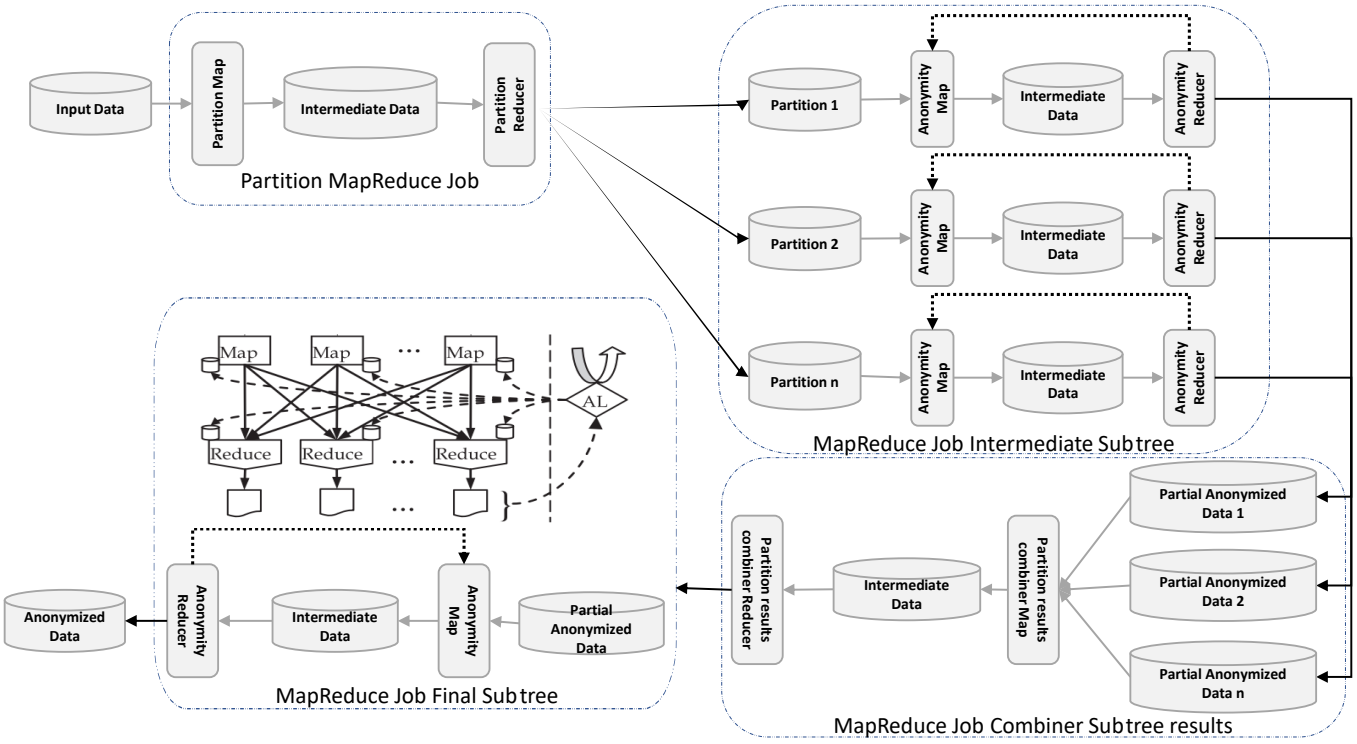

Figure 2. Subtree data-flow diagram in the MapReduce platform.

However, we identify the following architectural limitations of the MapReduce platform for implementing the subtree anonymization algorithm. These include the issues associated with a partition, memory, and iteration management. We argue that these limitations create execution complexity and performance degradation in various stages. We discuss these problems in detail in the following sections.

3.3.1. Partition

Processing data in MapReduce requires a map task to process a portion of the input data by assigning key-value pairs followed by generating intermediate data. The intermediate data is stored in a local disk of each executor node after applying the hash function. The hash applied to each partition ensures that the output of a map task is arranged using

the sort and shuffle process. This hash order ensures reducers can access their respective key-value pairs based on intermediate data locality [36].

An uneven hash partitioning of intermediate values may create skew data in multiple places. For instance, a node that contains a proportionally larger number of records than other nodes would result in tuple skewness [11]. As a consequence, a reducer coordinating these multiple nodes to process the outcomes now will have to wait for a significant time until the node containing the larger number of records completes. Similarly, key skewness [37] may happen when there is a big difference in the generalization levels being applied to different groups of attributes e.g., applying a single generalization level versus multiple generalization levels. This phenomenon would most likely happen more often when the k-group size is larger (e.g., there are more attributes).

To put more formally, let n be the number of tuples and m be the number of attributes in a dataset D, and let s and t represent the number of mapper and reducers, respectively. Then, mapper produces $m+1$ key-value pairs which yields $\mathcal{O}(1)$ space and $\mathcal{O}(m*n/s)$ time complexity [38]. However, the reducer yields $\mathcal{O}(1)$ and $\mathcal{O}(m/k*n/t)$ for space and time complexities respectively, where k denotes a k group size. The increase in s causes fewer n, which reduces the computing time for the mapper process, and by increasing the number of mappers (s), we get better big $\mathcal{O}$ complexity because $m*n$ is divided by the number of mappers [12,39].

3.3.2. Memory

As a mapper loads the input data from disk to memory (of the execution node), the results (i.e., intermediate data) are transferred and stored from the memory to the disk (of the same node). The reducer loads the intermediate data into the memory again (from the disk) of the execution node where the reducer runs on to process and subsequently store the results back to the disk [36]. Without the support of cache, any values that are produced in different stages (i.e., input, intermediate, or output) are stored in the disk and accessed each time the read/write of these values are required. This architectural design of MapReduce adds an excess overhead for I/O operations as well as demands for a larger storage capacity. We will put this more formally. Let subtree uses N_{It} for non-iterative jobs, and It for iterative jobs to convert dataset to anonymized data. J represents a MapReduce job, every J reads R times from the disk and W times writes on disk. I represents the number of iterations needed for each J. Then, I depends on multiple factors including the number of attributes, k group size, and generalization hierarchy. We use the following equation to calculate the total number of R and W operations in MapReduce subtree (ST).

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

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

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

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

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

where GL denotes generalization level in k-group.

3.3.3. Iteration

We argue that there are two architectural design principles of the MapReduce platform that create significant overheads for any iteration tasks. The first one is related to the I/O principle where any intermediate results have to be written to disk and subsequently read by the executor memory as we discussed in Section 3.3.2. The second one is related to the data locality principle where any data processing must be done on the same cluster node which holds the data to be processed. As a consequence, the result of the computation process also has to be saved in the same cluster node. The problem arises when the data read is required by other cluster nodes. In this case, the message exchange is required over the network which could cause noticeable delay and will be multiplied by each iteration process.

With the disk I/O-based operation and data locality principles, we argue that any algorithm that involves intensive iterations such as the subtree generalization can cause significant overheads at multiple places such as at *Disk I/O, Network*, and *Scheduling* [40].

Disk I/O overhead: Significant I/O overheads may occur at many different stages of subtree generalization where the stage involves an intensive iteration such as applying generalization levels for attributes, calculating privacy and utility scores, finding the most optimal generalization level, and re-applying the generalization based on the optimal generalization level.

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

Scheduling Synchronization Overhead: Assume a situation in which there are two mappers with different workloads and one mapper takes a significantly longer time to complete. In this case, the reducer processing the results of these two mappers needs to wait until both mappers complete their jobs. This is referred to as scheduling synchronization. However, if there are many mappers in iterations where the difference in the workloads are observed, the scheduling synchronization overhead can be increased as the number of imbalances across mappers happens.

4. Our Proposal

In this section, we provide a detailed description of our proposed approach for Spark-based subtree anonymization. Our proposal consists of three phases, where each phase output is required as an input for the next phase. We inherit the application of Spark Resilient Distributed Dataset (RDD) design and data partitioning mechanism for our approach and describe the step by step data flow to address the concerns discussed in Section 3.3. We robust our approach by segregating the computation of data anonymization using Figure 3 illustration. We discuss the details of each phase and the specific improvements we have made to resolve MapReduce-based issues in the following three phases:

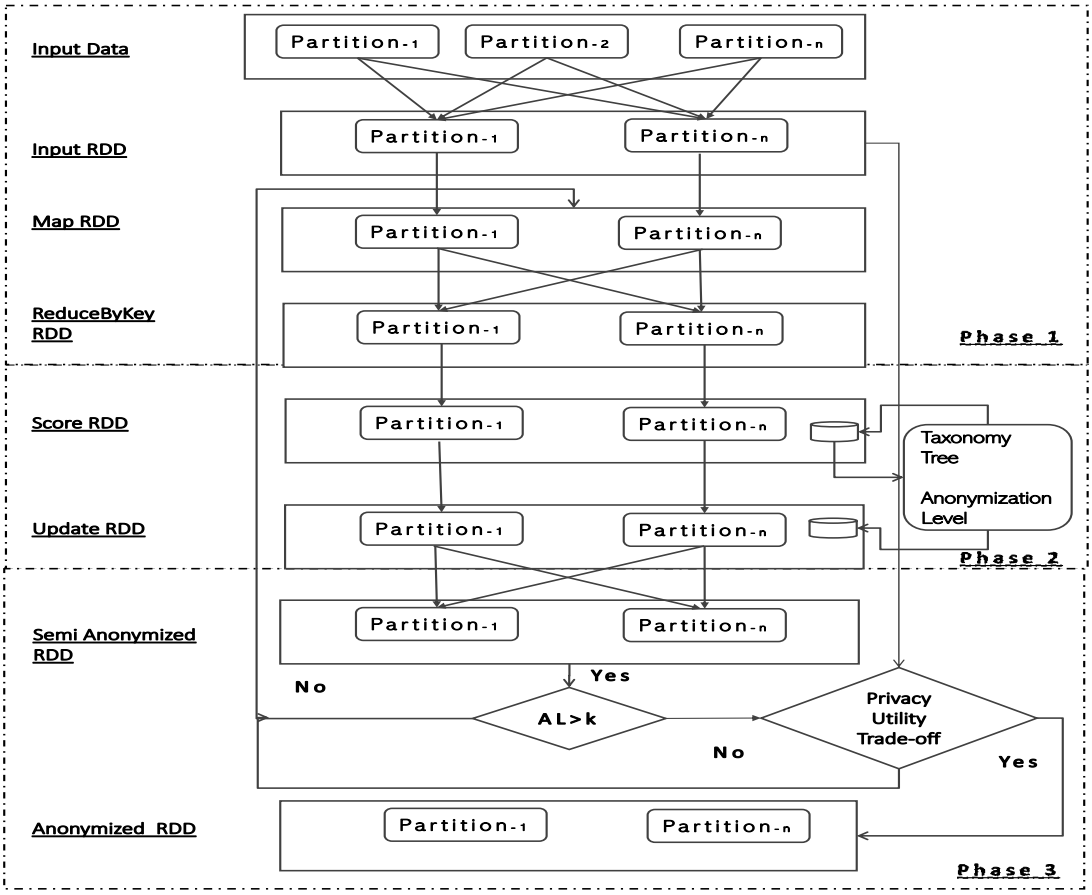

Figure 3. Proposed Spark subtree model.

4.1. Phase 1—Initialization

This phase ensures that each RDD partition contains the optimal number of records without duplication to provide a balanced workload of each partition. The original data records are counted and then assigned with a frequency value based on the times of appearance of that specific record in the whole dataset. we increase the stability by using the total record count approach to address both tuple and key skewness problems discussed in Section 3.3.1. In this phase, we provide new partition management that can avoid both tuple skewness and key skewness. The following steps detail our partition strategy. With roughly the equal number of records contained in each RDD partition, each partition executes in parallel by taking approximately a similar processing time.

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

We present this initialization step involving efficient partition in Algorithm 2. In Step 2, the "partition factor" indicates the variable that contains the number of records and the capability of the node. Step 3 uses Map_RDD to transform the input *RDD_in* as a key, and the value showing the key-value pairs used to process the data such as (r,C_r), where r represents records and C_r denoted the count in each Map. At this phase, the key-value pair is used, the key represents a single record while the value represents the number of times a key (a record) has appeared in the dataset. The ReduceByKey_RDD in Step 4 reads the Map_RDD key-value pairs (r,C_r) and aggregates the value for the same key. Then, the count of the same r is summed up together to find the $\sum C_r$ which represents the total number of record counts across all partitions. Please note that this process requires shuffling the data from different partitions in the executor nodes to exchange the values for the same key over the network.

Algorithm 2: Phase 1—Initialization Phase of Spark subtree.

Input: D
Output: (r, $\sum C_r$)
1 **begin**
2 $RDD\_in \leftarrow$ RDD (D, Partition factor)
3 Map_RDD (r,C_r) $\leftarrow$ Map (RDD_in)
4 ReduceByKey(r, $\sum C_r$) $\leftarrow$ ReduceByKey (Map_RDD, Partition factor)
5 **return** (r, $\sum C_r$)
6 **end**

4.2. Phase 2—Generalization

This phase calculates the privacy and utility scores for each attribute. The privacy and utility scores are used to find the most optimal generalization level to be applied for a certain attribute. Frequently referenced intermediate values (e.g., the privacy and utility scores and the results of the generalization level being applied) are stored first in memory and then moved to a cache to reduce any potential I/O overhead discussed in Sections 3.3.2 and 3.3.3. The purpose of this phase is to apply the most optimal generalization level according to the privacy and utility scores. The frequent use of memory and cache increases the robustness of our proposal. The memory holds the intermediate results for the computation of the privacy and utility scores of each attribute, the results of the generalization level are cached to avoid expensive disk access.

The generalization phases (Section 4.1) results as an input to compute the score value. We describe the details of this phase in Algorithm 3. Step 2 assigns A_v as child C_A in r for the generalization level for QID while P_A is assigned to all QID based on its C_A in TT. This process applies one level of generalization for one attribute (i.e., one iteration) and holds the results *in memory* so that the results can be used in the subsequent step. Steps 4–7 are used to compute the privacy and utility scores which are denoted as $Score_{ILPG}$ (QID) as following, based on [41,42].

$$Score_{ILPG}(QID) = \frac{IL(QID)}{PG(QID)+1}, \tag{2}$$

where $IL(QID)$ contains the result of information loss for QID while $PG(QID)$ contains the result of privacy gain for QID. The details of the calculation for $IL(QID)$ and $PG(QID)$ are depicted in Equations (3) and (4), as follows, respectively.

$$IL(QID) = En(P_A) - \sum_{C_A \in P_A} \left(\frac{|C_A|}{|P_A|} \right) En(C_A), \tag{3}$$

where $|C_A|$ represents the child attribute and $|P_A|$ represents the parent attribute for the given QID. $E_n(C_A)$ and $E_n(P_A)$ denote the entropy value of child and parent attributes respectively.

$$PG(QID) = A_{P_A(QID)} - A_{C_A(QID)}, \quad (4)$$

where $A_{P_A(QID)}$ and $A_{C_A(QID)}$ contain the Anonymization Level (AL) of the parent and child QIDs. Steps 7–10 are used to identify and update the best generalization level based on the privacy and utility score calculated in the earlier steps. The process goes through each r iterating over each A_v, where any A_v belonging to QID is considered to be *qid*. The A_v is considered to be C_A when the value is compared in TT. The C_A is compared with the same QID attributes in DOM. Once the C_A is found in TT, the C_A value is replaced by its P_A parent nodes. Then, the A_v values are replaced from C_A to P_A for each r to obtain r^*. Finally, the RDD returns anonymized key-value pairs (r^*,$\sum C_r$).

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

Algorithm 3: Phase 2—Generalization Phase of Spark subtree.

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

4.3. Phase 3—Validation

This phase validates if the generalized dataset meets the k-anonymization requirements, we provide a mechanism to deal with frequently referenced intermediate values (e.g., semi-anonymized dataset) by caching it to reduce the overheads discussed in Sections 3.3.2 and 3.3.3. In this phase, we validate if the full anonymization has been achieved i.e., the optimal generalization levels for all attributes have been applied up until they do not violate the k-anonymization constraint. This phase improves the intermediate results access time by storing semi-anonymized attributes into a memory cache to avoid expensive disk access and improve memory management.

Algorithm 4: Phase 3—Validation Phase of Spark subtree.

Input: $(r^*, \sum C_r)$,k
Output: (RDD^*)
1 **begin**
2 ReduceByKey $(r^*, \sum C_r^*) \leftarrow RDD\_update\big((r, \sum C_r), Partitionfactor\big)$
3 **if** $(AL \leq k)$ **then**
4 $(\sum C_r \leftarrow \sum C_r^*)$
5 $(r \leftarrow r^*)$
`/* return to Map_RDD in phase 1 if need more generalization */`
6 **return** $(r, \sum C_r)$
7 **else if** $(AL > k)$ **then**
8 **for** C_r^* *in* $\sum C_{r^*}$ **do**
9 $r^* \leftarrow (r^*, C_r^*)$
10 **return** (r^*)
11 **end**
`/* completely anonymized */`
12 Update $RDD^* \leftarrow r^*$
13 **return** RDD^*
14 **end**

This phase use the results of Section 4.3 as input for the final computation of the anonymization process. The detail of this phase is depicted in Algorithm 4. Step 2 is used to update the partition based on the Phase 1 strategy. Steps 3–6 are used to check whether $\sum C_r^*$ that contains the total number of generalized records (represented as AL) meets the k-anonymization constraint or not. If AL has fewer records than k-anonymization constraint, the semi-anonymized records $\sum C_r^*$ are required to be copied to a new partition of a map $\sum C_r$ and returns to Phase 1. Steps 7–11 are used in the case where the full generalization is achieved—that is, the number of generalized records meets k-anonymization constraint. Then, a key is assigned for each (distinct) fully generalized record where the value of a fully generalized record is used as a value. Finally, Step 12 saves all fully anonymized records to memory.

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

5. Experimental Results

In this section, we first describe our experimental setup including the details of the datasets and the system environment configurations. Subsequently, We compare our proposal with existing approaches based on record volume and number of records. We then provide the experimental results for our model on Adult and Irish datasets. we further investigate the impact of our proposal on memory and iteration performance. Finally, we discuss the results of the privacy and utility scores obtained through several privacy and utility measurement metrics.

5.1. Datasets

The experiments are carry out using two datasets: US Census dataset (i.e., Adult dataset) [34] and Irish Census dataset [43], larger datasets are created using the similar

proposed approach proposed [25] for the experiments. Tables 5 and 6 illustrates each quasi-identifiable attribute (QID) we used in our experiments and generalization level (GL) of each QID obtained from the taxonomy trees for Adult and Irish datasets. The sensitive attributes are set to the "Salary" in the Adult dataset and the "Industrial Group" in the Irish dataset.

Table 5. Adult dataset.

QID	Distinct Qid Value	*GL*
Age	74	3
Work Class	8	4
Education	16	4
Race	5	2
Gender	2	1
Native Country	41	3

Table 6. Irish dataset.

QID	Distinct Qid Value	*GL*
Age	70	4
Economic Status	9	2
Education	10	4
Marital Status	7	3
Gender	2	1
Field of Study	72	2

5.2. System Environment Configurations

We configured Yarn and Hadoop Distributed File System (HDFS) using Apache Ambari. The HDFS was used to distribute data in a NameNode (worked as a master node), a secondary NameNode, and six DataNodes (worked as worker nodes). We allocated 3 GB memory to Yarn NodeManager, and 1 GB memory to ResourceManager, Driver, and Executor memories each. We used Spark version 2.1 [44] along with Yarn as a cluster manager. The details of the experimental setup for both Spark platform and datasets are illustrated in Table 7.

Table 7. Spark and dataset setup parameters.

	Spark Setups					Dataset Setups			
	Driver Memory	**Executor Memory**	**No of Executors**	**Executor Cores**	**No of Partitions**	**k Group Size**	**Dataset Sizes/Records**	**Number of *QIDs***	**Datasets**
Figure 4	6.5 GB	2 GB	10	2	18–242	100	2 GB–30 GB	6	Adult
Figure 5	15 GB	3.5 GB	8	3	40	100	1.2 B	6	Adult
Figure 6	6.5 GB	4 GB	12	3	24	1000	0.1 B–1 B	6	Adult- Irish
Figure 7	6.5 GB	4 GB	12	3	24	50–1000	0.1 B–1 B	6	Adult
Figure 8	6.5 GB	4 GB	12	3	24	1000	0.1 B–1 B	2–6	Adult
Figure 9	6.5 GB	4 GB	12	3	24	1000	0.1 B	6	Adult
Figure 10	6.5 GB	4 GB	12	3	24	10,000	0.6 B	6	Adult
Figure 11	6.5 GB	4 GB	2–16	3	2–64	1000	0.8B	6	Adult
Figures 12–15	6.5 GB	4 GB	12	3	24	2–100	31062	6	Adult-Irish
Figure 16	6.5 GB	4 GB	12	3	24	100	31,062	6	Adult

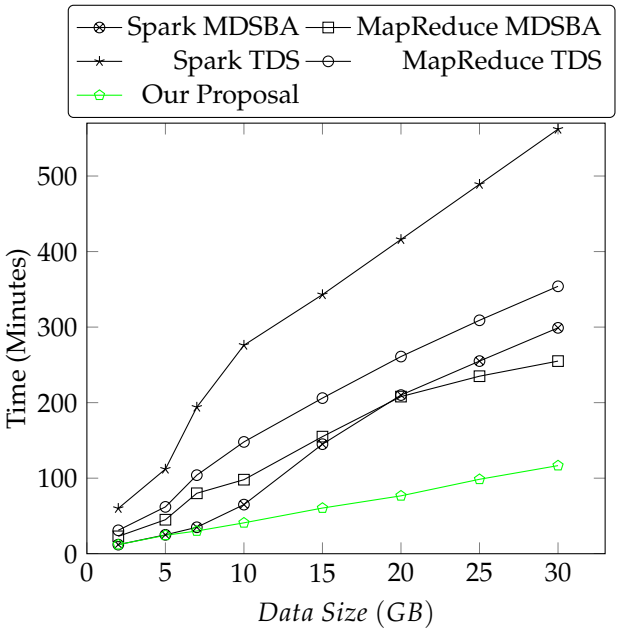

Figure 4. Performance comparison with existing subtree approaches based on increasing data size.

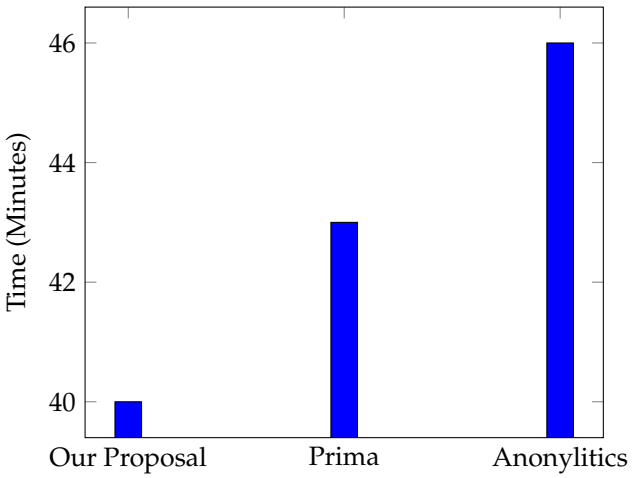

Figure 5. Performance comparison with existing spark based *k*-anonymity approaches.

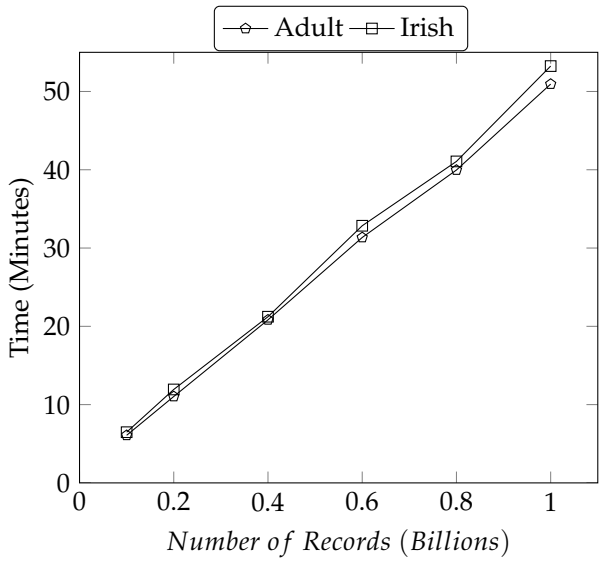

Figure 6. Scalability comparison for Adult and Irish dataset.

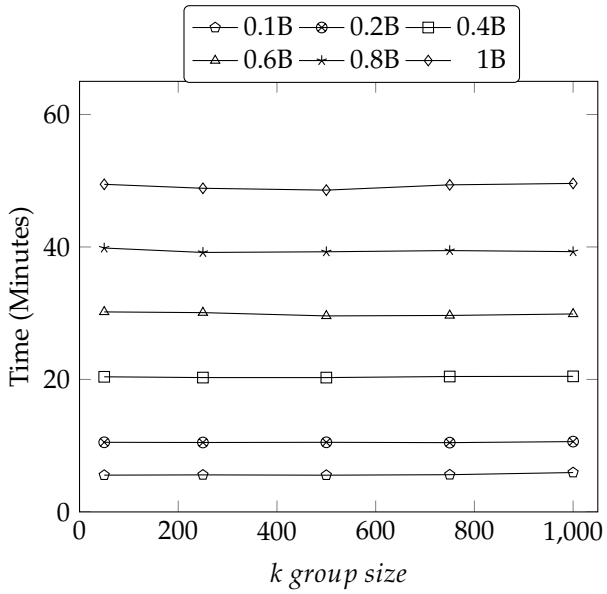

Figure 7. Performance against the # of records.

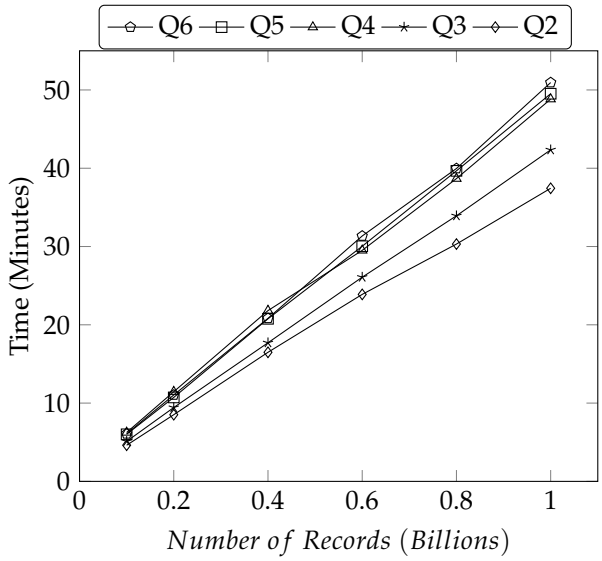

Figure 8. Scalability against the # of QID.

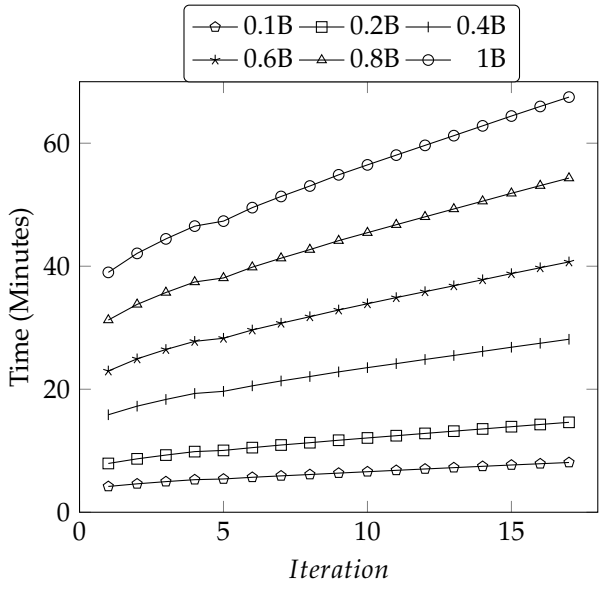

Figure 9. Performance of iteration against the # of records.

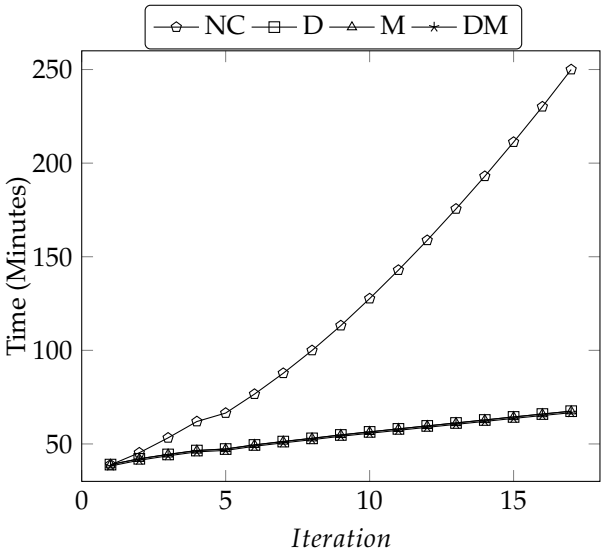

Figure 10. Performance of different cache management on iteration.

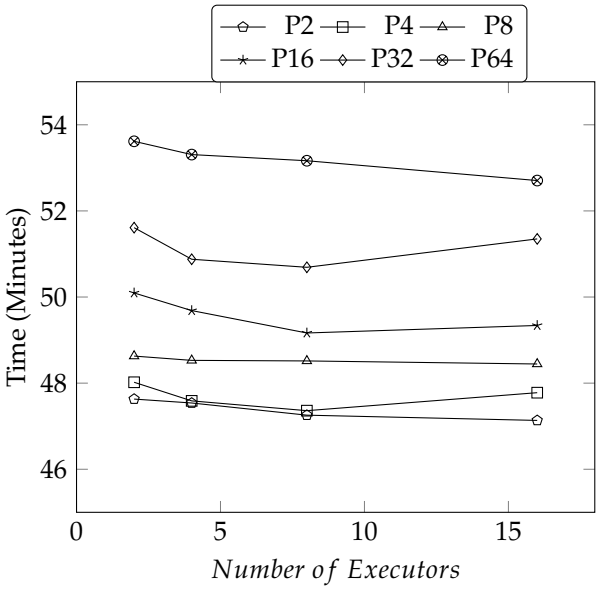

Figure 11. Performance against the # of partitions.

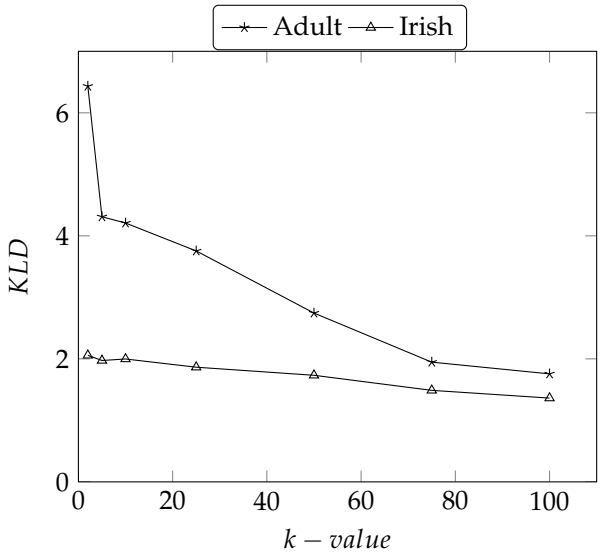

Figure 12. *KLD* Performance of privacy.

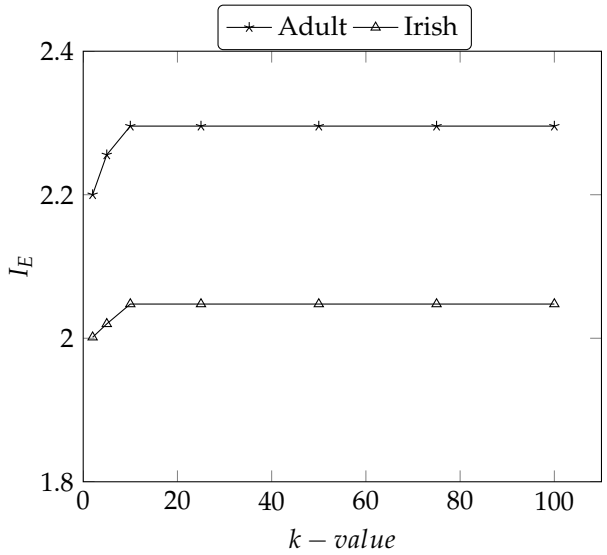

Figure 13. I_E Performance of privacy.

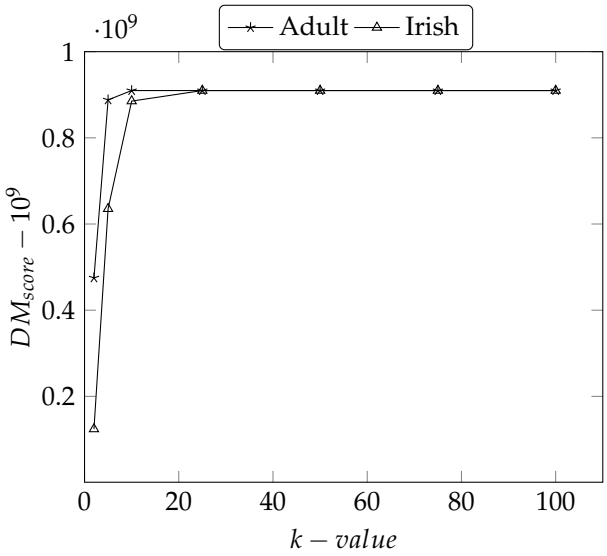

Figure 14. *DM* Performance of Utility.

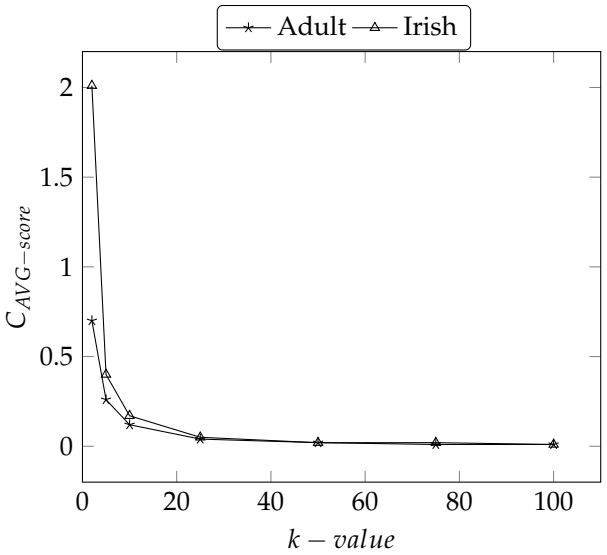

Figure 15. C_{AVG} Performance of Utility.

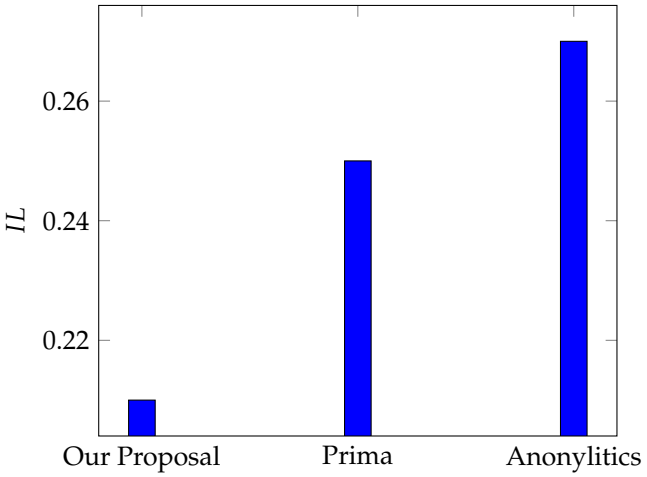

Figure 16. Information loss comparison between the existing approaches.

5.3. Performance and Scalability

We ran experiments to understand performance and scalability in terms of memory and iteration management. We ran our experiments 10 times and used the average value to ensure the reliability and consistency of the results. We ensured that the experiments for each dataset use a constant number of partition sizes (i.e., 24) instead of the default partition size. Fixing the partition size ensures that the data can be processed with an equal number of executors.

5.3.1. Performance Comparison with Existing Subtree Approaches

In this section, we discuss the results of our proposal in comparison with existing subtree MapReduce and Spark-based methods. We compare all the approaches based on the volume(size) of data. The increasing volume of data validates the requirements of big data, while the anonymity parameters validate the *k*-anonymity requirements [45].

We conduct the experiments to compare our approach with Spark and MapReduce multi-dimensional sensitivity-based anonymization for data (MDSBA) [21,22] against the growing size of data. Figure 4 compares the execution times of Spark MDSBA [22], MapReduce MDSBA [21], Spark Top-Down Specialization(TDS) [20], MapReduce TDS [11] with our proposal. The results show that our approach has the least amount of execution time in comparison with the other approaches. Moreover, we observed that the execution time increases linearly along with the increase in data size in all three approaches. Spark-based approaches such as our proposal and Spark MDSBA have almost the same performance when the data size is less than 10 GB, while when it comes to the bigger data size such as bigger than 10 GB, the execution time is much higher in other approaches in comparison with our approach.

5.3.2. Performance Comparison with Existing Spark-Based *k*-Anonymity Approaches

In the second set of results compare the performance of our approach with state-of-the-art Spark-based *k*-anonymity approaches with constant *k* group size, number of records size, and generalization level as shown in Table 7. Figure 5 compares the execution time across different Spark-based approaches such as Prima [24], Anonylitics [23]. The results indicate that our proposal yields the lowest execution time compared to the other platforms, while Anonylitics shows the highest execution time. We also identified that our approach uses a smaller number of RDDs and parallelism during the execution of each partition in its respective executors. Our approach measures the score and updates the anonymity in its prospective RDDs for all generalization levels of each QID. However, the Prima approach measures and updates the score of each leaf as a single RDD. Thus, the increments in the generalization level increased the number of leaves which caused more execution time for *k*-group size.

5.3.3. Performance Comparison on Adult and Irish Datasets

In this section, we perform an experiment to identify the impact and execution behavior of various datasets on our proposed model. We use Adult and Irish datasets for our performance experiment to understand the impact of execution time against the growing number of records on the fixed size of 5 QID attributes and 1000 *k* group size. As seen in Figure 6, the execution time changes as soon as the number of records is increased. The execution time linearly increases as the number of records increase in both datasets. We observe that our approach computes various datasets with different generalization levels and *k* group size; however, the increase in distance *qid* value and generalization level may increase the execution time i.e., that is the case with the Irish dataset for "Field of Study" QID. Although both adult and Irish datasets are used for the same number of records, *k* group size, and number of QIDs but the execution time increase in the Irish dataset with the increase in distance *qid* value and generalization level.

5.3.4. Memory Effects on Performance and Scalability

In this section, we discuss our results based on three aspects, including (i) performance in terms of the growing size of records, (ii) performance compared to other similar approaches, and finally (ii) scalability in terms of the increasing the number of attributes.

We first analyzed the performance implication of our approach by increasing the number of record sizes against different *k*-group size. The results in Figure 7 show the execution time based on the increasing record size starting from 0.1 billion (10^8 records) to 1 billion (10^9 records). We observed that the execution time has a linear growth with respect to increasing dataset size. We also did not observe any distortion caused by *k*-group size as the execution times remain almost constant even though *k*-group size increases. We identified that this effect is because of two reasons: (i) The records are required for the measurement of privacy and utility score from RDD rather than the complete data records; thus, after each generalization step, the same records are aggregated and represented with the key-value pairs. The key-value pairs contain enough information and do not require additional calculation, (ii) Our anonymization process uses a broadcast mechanism that works as a data-sharing mechanism across executors which effectively reduces network I/O and memory, and disk I/O. Consequently, it reduces the computation time significantly.

The scalability of the distributed anonymization is benchmarked against the increasing QID size and is represented in Figure 8. We increased the adult dataset QID size with respect to increasing the number of record sizes. We discovered that the execution time is dependent on the size of QID and the variety of each *qid* value (i.e., the level of generalization applied). Thus, the higher size of QID set and diverse *qid* value cause the higher execution time. We observed that for the higher size of QID, the larger size of equivalence classes was needed to satisfy the *k*-anonymity requirements as it allowed a greater number of attributes grouped/partitioned together, thus it reduced the number of required iterations.

5.3.5. Iteration Effects on Scalability

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

Figure 9 compares the number of iterations against the execution time for various dataset sizes. We can observe that having more iteration leads to more execution time. When we increased the record size from 0.1 B and 0.2 B, we observed that the executor memory had enough space to accommodate the records while processing the anonymization. Thus, it does not invoke evacuation of memory due to overload. While as we increase the size of the record, the executor memory starts the evacuation process. It is noticeable that although each RDD is allocated with the same or a smaller number of input data, the anonymization process adds more data to the memory for execution. Thus, the larger the record size, the more records need to be evacuated to make enough space for execution.

Figure 10 compares the cache and Non-Cache (NC) effects on increasing generalization level on each iteration. We observed that the execution time for NC RDD has a higher execution time in comparison with cached RDD regardless of the storage levels such as DISK_ONLY (D) MEMORY_ONLY (M), or MEMORY_AND_DISK (DM). For the smaller dataset, it has more space to hold the cached RDD in memory. However, by increasing the dataset size, the RDD partitions need to be deleted from the memory and calculated again for the next transformation. In each iteration, RDD data needs to be scanned for finding the optimal generalization. To achieve this, more frequent visits to RDD data in memory were necessary thus increasing the execution time.

While we observed that DISK, MEMORY, and their combination provide a similar execution time, all these three storage options have different approaches for accommodating cache results. As described by [14,46], the combination of memory and storage is the most cost-effective operation for iteration by ensuring faster computation. We also observed that

after each iteration execution, the read and write time taken by memory and disk is slightly increased without recomputing the space and size for the next iteration.

Furthermore, we investigated the impact of RDD partition on the number of executors to identify a balance between high parallelism and using the available resources to the maximum capacity. A partition against the executor trade-off has been discussed in [20,47]. Having considered the results demonstrated in Figure 11, we can observe that the increase in the number of partitions improves the execution time as 64 partitions (denoted as P64) has more execution time in comparison with when only two partitions are used (P2). This means that the partition size and the executor number needs to be in balance to avoid any potential latency.

5.4. Privacy & Utility Trade-Off

We used the privacy and utility metrics for the measurement and validation of our proposed method. The data anonymization technique uses trade-offs between privacy and utility to quantify the success of an anonymization algorithm. A privacy level is estimated by recognizing the uniqueness of information, a low privacy normally implies that it is anything but difficult to distinguish an individual (a tuple or record) from a group (e.g., numerous records). We used two privacy metrics Kullback-Leibler−divergence (KLD) and Information Entropy (I_E) to evaluate the impact of the privacy level of our proposal.

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

5.4.1. Kullback-Leibler-Divergence (KLD)

KLD measures the likelihood of the presence of the original attribute in the anonymized attribute for each record [48]. For example, let the original attribute of the Job is "Writer" and is anonymized into "Artist". The KLD measures the possibility of guessing the original data of "Writer" from "Artist".

In our approach, we calculate KLD on the final anonymized dataset by measuring the likelihood of the presence of each attribute and sums all the value for each attribute within a record and repeat this for all records.

KLD can be computed based on the formula represented in Equation (5).

$$KLD = \sum_{r=1}^{n} P_{RDD^*}(r) \log \frac{P_{RDD^*}(r)}{P_{RDD\_in}(r)} \tag{5}$$

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

Figure 12 presents the KLD values of our proposed subtree generalization implementation on Adult and Irish datasets. The KLD values increase with the increase of k-group size and are very close to the comparative approaches discussed [25,49].

The results of KLD metric on the Adult and Irish datasets are shown in Figure 12. The KLD values only increased from around k group size 2 to 5. After k-value (i.e., group size) = 5 the KLD values remain the same for the rest of the k group size. The visible increase of KLD from k-value 2 to 5 (and slight changes from 5 onward) is due to the active generalization level being applied. At approximately k-value 10, all generalization has been applied and there are no more changes to the rest of the k-value thus KLD value remains identical. The overall observation of the changes in KLD values is similar to that of the Adult dataset. However, we observe that the average KLD values are much higher in the Irish dataset than in the Adult dataset. This is due that the Irish dataset has more generalization levels for each QID which increases the chances of more number of QIDs sharing the same value.

5.4.2. Information Entropy (I_E)

I_E is used to measure the degree of how uncertain it is to identify the original value from the anonymized value within a QID set [50]. The entropy value of I_E is 1 if all the qid attributes are identical in the anonymized dataset for the same QID. To compute $I_E(QID)$, (1) the likelihood of the presence of the original attribute in a record is calculated, (2) sums up the value of (1) for each attribute in a record (denoted as $P_{RDD^*(qid)}$), (3) continues (1) and (2) for each QID, (4) sums up the value of (3) for all records. Please note that if all attributes are changed between the original record and the anonymized record, the value of P_{RDD^*} is 1.

$$I_E = -\sum_{qid=1}^{n} P_{RDD^*(qid)} \log P_{RDD^*(qid)} \tag{6}$$

Based on Equation (6), we computed $I_E(QID)$ for single QID. To obtain the I_E for the entire anonymized dataset (denoted as RDD^*), we calculated the I_E for RDD^* by taking the average of all QID. The entropy value of I_E is 0 if the records are identical between the original dataset and the anonymized dataset within the matching equivalent class. The maximum value of I_E is achieved when the original record sets are very different from the anonymized record sets for a given QID. A higher value of I_E represents more uncertainty (i.e., higher privacy).

Figure 13 shows the privacy level in terms of the entropy of our proposed approach. Generally, the entropy increases with k size. Though the I_E score is highest at ($log_2 k$) in [51], the I_E score of our proposal works better than the scheme proposed by [29]. The performance of our approach is close to the higher end of standard and achieves much higher privacy levels achieved by the scheme proposed by [25]. As the entropy represents the information content of data change, the entropy after data anonymization should be higher than the entropy before the anonymization which is the phenomenon observed in our results.

The results of I_E metric on Adult and Irish dataset are shown in Figure 13. Again, the values between the Adult and Irish remain in the study parallel which ensures that the implementation of our data anonymization technique in Spark did not destroy the privacy level. The average of I_E values in the Adult dataset is lower compared to the Irish dataset. Our investigation reveals that the Adult dataset contains a relatively small number of different QIDs that share the same value as the result of anonymization. The smaller k value affects the I_E value more compared to the greater k value due to the number of the same values in QID attributes. This affects the higher I_E value as it is easier to identify a unique record within the same equivalent class compared to the Irish dataset which has a larger number of different QIDs that share the same value.

5.4.3. Discernibility Metric (DM)

DM reports the data quality resulting from the degree of data degradation due to data anonymization based on a tuple within an equivalent class (EC). Let EC be the set of equivalence classes of a k-anonymized dataset RDD^*. EC_i is one of the equivalence classes of $\mid EC \mid$. The DM metric can be expressed more formally as Equation (7).

$$DM_{score} = \sum_{EC_i \in RDD^*} \mid EC_i \mid^2, \tag{7}$$

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

Discernability Metrics (DM) [52] measure the cardinality (i.e., distinctness) of the equivalence class. For a low group size of k, the cardinality of equivalence is too small. If the privacy level is high (e.g., a higher group size of k), the discernability metric increases sharply which increases the cardinality of an equivalence class. Equivalence classes with a

large cardinality tend to group datasets in a large range leading to large information loss. Figure 14 presents the discernability penalty of 30,162 records for adult dataset.

We observed that the overall trends for the DM to the DM values observed in other similar approaches in [25,51]. As the k-group size increases, more records are part of an EC, and thus records are less distinguishable from each other. The Irish dataset shows higher DM scores compared to the Adult dataset because there are more chances to make on tuples on the Irish dataset as it contains more distinct qid values. For both datasets, the DM score stays steady which shows low sensitivity to the growth of k-group size (e.g., no more changes in EC).

5.4.4. Average Equivalence Class Size Metric (C_{AVG})

C_{AVG} is used to measure data utility based on attributes of the average size of the equivalence class. The increase in the number of equivalence sizes results in a higher data utility as it is more difficult to identify an attribute among many identical attributes. In the k-anonymized dataset, the size of the equivalence classes is greater than or equal to k. As a result, the quality of the data is lower if the size of all or part of the equivalence classes greatly exceeds the value of k. The score of C_{AVG} is sensitive to the k-group size [53]. C_{AVG} for RDD^* is calculated as Equation (8).

$$C_{AVG} = \frac{|RDD^*|}{|EC|}/k \tag{8}$$

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

Figure 15 represents the results of C_{AVE} for increasing group size of k. The decreasing score value against the increasing size of k is observed indicating that the size of the created ECs is equal to the given k, that is the ECs contain the number of generalized records that satisfy the k-anonymity. As the value of k increases, the EC has more records than the k requirement due to higher generalization level, this keeps increasing the C_{AVE} score value.

The trend of C_{AVG} scores was similar to DM as both metrics were based on the calculation according to the size of equivalence classes on the number of records in the dataset. The comparison of C_{AVG} and DM scores are very hard to compare the values for Adult and Irish different datasets. At the time of C_{AVG} consider the number of records, however, the DM score does not take into account the records of the dataset [54]. As defined earlier the equivalent class EC contains the identical number of QID attributes in a table, the increase in k will increase the qid for each EC class. In Figure 14 score for the Irish dataset is significantly different from the Adult dataset, this is because the Adult dataset applies generalization on "Race" QID, whereas Irish use "Field of Study" for the highest generalization. We observe the number of distance qid values for "Race" and "Field of Study" QID are large in the margin as shown in Tables 5 and 6 respectively. The increase in k increase the GL accordingly, with the "Field of Study" QID in Irish dataset changes a large number of the attribute that increases the EC which impact the C_{AVG} and DM score compared to Adult dataset "Race". In addition to that, there are 16 GL in Irish data and 17 in Adult data, once all the GL are applied for $k > 20$ we observe both datasets returns the same utility as observed for the C_{AVG} and DM scores for $k > 15$ in Figures 14 and 15.

5.4.5. Information Loss (IL)

The Information Loss in Equation (9) is calculated with method used in [18] that computes the amount of information loss of information after generalizing a particular value to a general value in the anonymization process. Let the record r and qid be the number of attributes, the $U_{n.m}$ and $L_{n.m}$ are the upper and lower bounds of the n^{th} qid

in the m^{th} r for anonymized data while the maximum and minimum for m attributes is denoted as MAX_m and MIN_m in the original dataset.

$$IL = \frac{1}{r.qid} \sum_{n=1}^{r} \sum_{m=1}^{qid} \frac{|U_{n.m}| - |L_{n.m}|}{|MAX_m - MIN_m|} \tag{9}$$

Anonymization via generalization and/or suppression is able to protect the privacy of individuals, but at the cost of information loss especially for high-dimensional data. We compare the results of our approach with the existing state-of-the-art approaches for IL such as Prima [24], and Anonylitics [23].

The results in Figure 16 shows that our approach provides less information loss compared to Prima and Anonylitics. We use a constant k size with a fixed number of records. We identify the using Score and update and semi anonymized RDD (Section 4), our approach reduces the duplication of records and anonymizing the duplicate records by replacing *qid* as single records. The equivalence anonymization for the same records normalized the information loss thus providing minimum information loss.

6. Conclusions and Future Work

This study proposes a generic framework for implementing subtree-based generations on Apache Spark. Our proposal is implemented using a series of RDDs that support more efficient partition management, improves memory usage that also uses cache to store frequently referenced values, and support a better iteration process which is much more suited for an iteration-intensive algorithm such as subtree generalization. Our proposed approach outperforms the existing similar approaches on various data sizes and k group sizes. Our proposal not only reduces the complexity of the operation and improves the performance but also shows high data utility scores while maintaining a competitive level of data privacy required for any data anonymization techniques. We plan to extend our study by further exploring the suitability of other data anonymization approaches for the Apache Spark platform. For instance, we plan to investigate one of the multi-dimensional data anonymization strategies such as Mondrian [53] to examine the support for recursive operations in Apache Spark.

Author Contributions: Conceptualization, S.U.B. and J.J.-J.; methodology, S.U.B.; software, S.U.B.; validation, S.U.B. and J.J.-J.; formal analysis, S.U.B. and H.A.; investigation, S.U.B.; resources, S.U.B. and J.J.-J.; writing—original draft preparation, S.U.B.; writing—review and editing, J.J.-J. and H.A.; visualization, S.U.B. and H.A.; supervision, J.J.-J.; project administration, S.U.B. and J.J.-J. All authors have read and agreed to the published version of the manuscript.

Funding: This work was made possible by the support of the grant (MAUX1912) from the Ministry of Business, Innovation, and Employment (MBIE) of the New Zealand Government.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Yu, S. Big privacy: Challenges and opportunities of privacy study in the age of big data. *IEEE Access* **2016**, *4*, 2751–2763. [CrossRef]
2. Jang-Jaccard, J.; Nepal, S. A survey of emerging threats in cybersecurity. *J. Comput. Syst. Sci.* **2014**, *80*, 973–993. [CrossRef]
3. Baryalai, M.; Jang-Jaccard, J.; Liu, D. Towards privacy-preserving classification in neural networks. In Proceedings of the 2016 14th Annual Conference on Privacy, Security and Trust (PST), Auckland, New Zealand, 12–14 December 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 392–399.
4. Fung, B.C.; Wang, K.; Wang, L.; Hung, P.C. Privacy-preserving data publishing for cluster analysis. *Data Knowl. Eng.* **2009**, *68*, 552–575. [CrossRef]
5. Fung, B.C.; Wang, K.; Wang, L.; Debbabi, M. A framework for privacy-preserving cluster analysis. In Proceedings of the 2008 IEEE International Conference on Intelligence and Security Informatics, Taipei, Taiwan, 17–20 June 2008; IEEE: Piscataway, NJ, USA, 2008; pp. 46–51.

6. Fung, B.C.; Wang, K.; Yu, P.S. Top-down specialization for information and privacy preservation. In Proceedings of the 21st International Conference on Data Engineering (ICDE'05), Tokyo, Japan, 5–8 April 2005; IEEE: Piscataway, NJ, USA, 2005; pp. 205–216.
7. Zhang, X.; Liu, C.; Nepal, S.; Yang, C.; Dou, W.; Chen, J. SaC-FRAPP: A scalable and cost-effective framework for privacy preservation over big data on cloud. *Concurr. Comput. Pract. Exp.* **2013**, *25*, 2561–2576. [CrossRef]
8. Jain, P.; Gyanchandani, M.; Khare, N. Big data privacy: A technological perspective and review. *J. Big Data* **2016**, *3*, 25. [CrossRef]
9. Bazai, S.U.; Jang-Jaccard, J.; Zhang, X. A privacy preserving platform for mapreduce. In Proceedings of the International Conference on Applications and Techniques in Information Security, Auckland, New Zealand, 6–7 July 2017; Springer: Berlin, Germany, 2017; pp. 88–99.
10. Bazai, S.U.; Jang-Jaccard, J.; Wang, R. Anonymizing k-NN classification on MapReduce. In Proceedings of the International Conference on Mobile Networks and Management, Melbourne, Australia, 13–15 December 2017; Springer: Berlin, Germany, 2017; pp. 364–377.
11. Zhang, X.; Yang, L.T.; Liu, C.; Chen, J. A scalable two-phase top-down specialization approach for data anonymization using MapReduce on cloud. *IEEE Trans. Parallel Distrib. Syst.* **2014**, *25*, 363–373. [CrossRef]
12. Zhang, X.; Liu, C.; Nepal, S.; Yang, C.; Dou, W.; Chen, J. Combining top-down and bottom-up: Scalable sub-tree anonymization over big data using MapReduce on cloud. In Proceedings of the 2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, Melbourne, Australia, 16–18 July 2013; IEEE: Piscataway, NJ, USA, 2013; pp. 501–508.
13. Zhang, X.; Nepal, S.; Yang, C.; Dou, W.; Chen, J. A hybrid approach for scalable sub-tree anonymization over big data using MapReduce on cloud. *J. Comput. Syst. Sci.* **2014**, *80*, 1008–1020. [CrossRef]
14. Shi, J.; Qiu, Y.; Minhas, U.F.; Jiao, L.; Wang, C.; Reinwald, B.; Özcan, F. Clash of the titans: MapReduce vs. spark for large scale data analytics. *Proc. Vldb Endow.* **2015**, *8*, 2110–2121. [CrossRef]
15. Maillo, J.; Ramírez, S.; Triguero, I.; Herrera, F. kNN-IS: An Iterative Spark-based design of the k-Nearest Neighbors classifier for big data. *Knowl.-Based Syst.* **2017**, *117*, 3–15. [CrossRef]
16. Zaharia, M.; Chowdhury, M.; Das, T.; Dave, A.; Ma, J.; McCauley, M.; Franklin, M.J.; Shenker, S.; Stoica, I. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, San Jose, CA, USA, 25–27 April 2012; USENIX Association: Berkeley, CA, USA, 2012; p. 2.
17. Bazai, S.U.; Jang-Jaccard, J. SparkDA: RDD-Based High-Performance Data Anonymization Technique for Spark Platform. In Proceedings of the International Conference on Network and System Security, Sapporo, Japan, 15–18 December 2019; Springer: Berlin, Germany, 2019; pp. 646–662.
18. Ashkouti, F.; Sheikhahmadi, A. DI-Mondrian: Distributed improved Mondrian for satisfaction of the L-diversity privacy model using Apache Spark. *Inf. Sci.* **2021**, *546*, 1–24. [CrossRef]
19. Zaharia, M.; Xin, R.S.; Wendell, P.; Das, T.; Armbrust, M.; Dave, A.; Meng, X.; Rosen, J.; Venkataraman, S.; Franklin, M.J.; et al. Apache spark: A unified engine for big data processing. *Commun. Acm* **2016**, *59*, 56–65. [CrossRef]
20. Sopaoglu, U.; Abul, O. A top-down k-anonymization implementation for apache spark. In Proceedings of the 2017 IEEE International Conference on Big Data (Big Data), Boston, MA, USA, 11–14 December 2017; pp. 4513–4521.
21. Al-Zobbi, M.; Shahrestani, S.; Ruan, C. Improving MapReduce privacy by implementing multi-dimensional sensitivity-based anonymization. *J. Big Data* **2017**, *4*, 45. [CrossRef]
22. Al-Zobbi, M.; Shahrestani, S.; Ruan, C. Experimenting sensitivity-based anonymization framework in apache spark. *J. Big Data* **2018**, *5*, 38. [CrossRef]
23. Pomares-Quimbaya, A.; Sierra-Múnera, A.; Mendoza-Mendoza, J.; Malaver-Moreno, J.; Carvajal, H.; Moncayo, V. Anonylitics: From a Small Data to a Big Data Anonymization System for Analytical Projects. In Proceedings of the 21st International Conference on Enterprise Information Systems, Crete, Greece, 3–5 May 2019; pp. 61–71.
24. Antonatos, S.; Braghin, S.; Holohan, N.; Gkoufas, Y.; Mac Aonghusa, P. PRIMA: An End-to-End Framework for Privacy at Scale. In Proceedings of the 2018 IEEE 34th International Conference on Data Engineering (ICDE), Paris, France, 16–19 April 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 1531–1542.
25. Bazai, S.U.; Jang-Jaccard, J. In-Memory Data Anonymization Using Scalable and High Performance RDD Design. *Electronics* **2020**, *9*, 1732. [CrossRef]
26. Gao, Z.Q.; Zhang, L.J. DPHKMS: An efficient hybrid clustering preserving differential privacy in spark. In *Proceedings of the International Conference on Emerging Internetworking, Data & Web Technologies*; Springer: Berlin, Germany, 2017; pp. 367–377.
27. Peethambaran, G.; Naikodi, C.; Suresh, L. An Ensemble Learning Approach for Privacy–Quality–Efficiency Trade-Off in Data Analytics. In Proceedings of the 2020 International Conference on Smart Electronics and Communication (ICOSEC), Tamilnadu, India, 10–12 September 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 228–235.
28. Jebali, A.; Sassi, S.; Jemai, A. Secure data outsourcing in presence of the inference problem: Issues and directions. *J. Inf. Telecommun.* **2020**, *5*, 16–34. [CrossRef]
29. Chakravorty, A.; Rong, C.; Jayaram, K.; Tao, S. Scalable, Efficient Anonymization with INCOGNITO-Framework & Algorithm. In Proceedings of the 2017 IEEE International Congress on Big Data (BigData Congress), Honolulu, HI, USA, 25–30 June 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 39–48.

30. Dwork, C. Differential Privacy. In *ICALP LNCS*; Bugliesi, M., Preneel, B., Sassone, V., Wegener, I., Eds.; Springer: Berlin, Germany, 2006; pp. 1–12.
31. Gao, Z.; Sun, Y.; Cui, X.; Wang, Y.; Duan, Y.; Wang, X.A. Privacy-preserving hybrid K-means. *Int. J. Data Warehous. Min. (IJDWM)* **2018**, *14*, 1–17. [CrossRef]
32. Yin, S.L.; Liu, J. A k-means approach for mapreduce model and social network privacy protection. *J. Inf. Hiding Multimed. Signal Process.* **2016**, *7*, 1215–1221.
33. Smith, C.; Albarghouthi, A. Synthesizing differentially private programs. *Proc. ACM Program. Lang.* **2019**, *3*, 1–29. [CrossRef]
34. Asuncion, A.; Newman, D. *UCI Machine Learning Repository*. 2007. Available online: https://archive.ics.uci.edu/ml/about.html (accessed on 2 March 2021)..
35. Sweeney, L. k-anonymity: A model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl. Based Syst.* **2002**, *10*, 557–570. [CrossRef]
36. Dean, J.; Ghemawat, S. MapReduce: Simplified data processing on large clusters. *Commun. Acm* **2008**, *51*, 107–113. [CrossRef]
37. Gufler, B.; Augsten, N.; Reiser, A.; Kemper, A. Handling Data Skew in MapReduce. *Closer* **2011**, *11*, 574–583.
38. Tao, Y.; Lin, W.; Xiao, X. Minimal mapreduce algorithms. In Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, New York, NY, USA, 22–27 June 2013; pp. 529–540.
39. Zhang, X.; Dou, W.; Pei, J.; Nepal, S.; Yang, C.; Liu, C.; Chen, J. Proximity-aware local-recoding anonymization with mapreduce for scalable big data privacy preservation in cloud. *IEEE Trans. Comput.* **2014**, *64*, 2293–2307. [CrossRef]
40. Kang, M.; Lee, J.G. An experimental analysis of limitations of MapReduce for iterative algorithms on Spark. *Clust. Comput.* **2017**, *20*, 3593–3604. [CrossRef]
41. Fung, B.C.; Wang, K.; Philip, S.Y. Anonymizing classification data for privacy preservation. *IEEE Trans. Knowl. Data Eng.* **2007**, *19*, 711–725. [CrossRef]
42. Fung, B.C.; Wang, K.; Chen, R.; Yu, P.S. Privacy-preserving data publishing: A survey of recent developments. *ACM Comput. Surv. (Csur)* **2010**, *42*, 1–53. [CrossRef]
43. *Central Statistics Office (Internet)*. 2011. Available online: https://www.cso.ie/en/census/census2011reports/ (accessed on 2 March 2021).
44. Spark Overview. In *Overview—Spark 2.1.0 Documentation*. Available online: https://spark.apache.org/docs/2.1.0/ (accessed on 2 March 2021)
45. Mehta, B.B.; Rao, U.P. Improved l-diversity: Scalable anonymization approach for privacy preserving big data publishing. *J. King Saud Univ. Comput. Inf. Sci.* **2019**. [CrossRef]
46. Gopalani, S.; Arora, R. Comparing apache spark and map reduce with performance analysis using k-means. *Int. J. Comput. Appl.* **2015**, *113*, 8–11. [CrossRef]
47. Zaharia, M.; Chowdhury, M.; Franklin, M.J.; Shenker, S.; Stoica, I. Spark: Cluster computing with working sets. *HotCloud* **2010**, *10*, 95.
48. Kifer, D.; Gehrke, J. Injecting utility into anonymized datasets. In Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, Chicago, IL, USA, 27–29 June 2006; ACM: New York, NY, USA, 2006; pp. 217–228.
49. Ayala-Rivera, V.; McDonagh, P.; Cerqueus, T.; Murphy, L. A systematic comparison and evaluation of k-anonymization algorithms for practitioners. *Trans. Data Priv.* **2014**, *7*, 337–370.
50. Ashwin, M.; Daniel, K.; Johannes, G.; Muthuramakrishnan, V. l-diversity: Privacy beyond k-anonymity. *ACM Trans. Knowl. Discov. Data* **2007**, *1*, 1–52.
51. Niu, B.; Li, Q.; Zhu, X.; Cao, G.; Li, H. Achieving k-anonymity in privacy-aware location-based services. In Proceedings of the IEEE INFOCOM 2014-IEEE Conference on Computer Communications, Toronto, ON, Canada, 27 April–2 May 2014; IEEE: Piscataway, NJ, USA, 2014; pp. 754–762.
52. Chakravorty, A.; Wlodarczyk, T.W.; Rong, C. A Scalable K-Anonymization solution for preserving privacy in an Aging-in-Place welfare Intercloud. In Proceedings of the 2014 IEEE International Conference on Cloud Engineering, Boston, MA, USA, 11–14 March 2014; IEEE: Piscataway, NJ, USA, 2014; pp. 424–431.
53. LeFevre, K.; DeWitt, D.J.; Ramakrishnan, R. Mondrian multidimensional k-anonymity. In Proceedings of the 22nd International Conference on Data Engineering (ICDE'06), Atlanta, GA, USA, 3–7 April 2006; IEEE: Piscataway, NJ, USA, 2006; p. 25.
54. Li, N.; Li, T.; Venkatasubramanian, S. t-Closeness: Privacy Beyond k-Anonymity and l-Diversity. In Proceedings of the 2007 IEEE 23rd International Conference on Data Engineering, Istanbul, Turkey, 11–15 April 2007; pp. 106–115.