SparkDA: RDD-Based High-Performance Data Anonymization Technique for Spark Platform

Sibghat Ullah Bazai(✉) and Julian Jang-Jaccard

Massey University, Auckland, New Zealand
{s.bazai,j.jang-jaccard}@massey.ac.nz

Abstract. Recent proposals in data anonymization have mostly been focused around MapReduce, though the advantages of Spark have been well documented. To address this concern, we propose a new novel data anonymization technique for Apache Spark. SparkDA, our proposal, takes the full advantages of innovative Spark features, such as better partition control, in-memory process, and cache management for iterative operations, while providing high data utility with privacy. These are achieved by proposing data anonymization algorithms through Spark's Resilient Distributed Dataset (RDD). Our data anonymization algorithms are implemented at two main data processing RDD transformations, FlatMapRDD and ReduceByKeyRDD, respectively. Our experimental results show that our proposed approach provides required data privacy and utility levels while providing scalability with high-performance that are essential to many large datasets.

Keywords: High-performance · Data anonymization · Spark · Big data mining · Privacy and utility

1 Introduction

With the popularity of Big Data, distributed parallel processing platforms have emerged with the features required for processing large amount of data for example high capacity storage, processing units, and execution memory, etc. Many traditional data anonymization techniques have been adapted to work along with such distributed parallel processing platforms, such as MapReduce, to take the advantages of many scalability features offered by them [7–9]. However, existing MapReduce-based data anonymization approaches would often suffer performance issues due to inherent MapReduce architectural design, which requires data (and their by-products such as intermediate data) to travel to disks frequently [17], the lack of mechanisms which can share data across multiple nodes or run iterative tasks more efficiently [10].

Spark [16] has been emerged as the next generation big data processing platform offering new advanced features that were limited in MapReduce [12]. With

J. K. Liu and X. Huang (Eds.): NSS 2019, LNCS 11928, pp. 646–662, 2019.
https://doi.org/10.1007/978-3-030-36938-5_40

the popularity of Spark in recent years, the proposals for data anonymization techniques to run on Spark platform also has emerged [2,3,13]. However, these proposals are often two sketchy to understand neither the details of the data anonymization strategies nor provide benchmark figures for privacy, scalability and performance. We present a novel data anonymization approach named SparkDA that implements data anonymization algorithms while taking the full advantages of Spark's advanced features. Our SparkDA provides the following capabilities.

- The main data anonymization algorithms are offered through Resilient Distributed Dataset (RDD) transformation, by designing FlatMapRDD and ReduceByKeyRDD transformations, respectively.
- By utilizing RDDS, our data anonymization algorithms are run in memory (instead of disk as done in MapReduce). This reduces the overheads of having to travel to expensive disk I/O – especially for intermediary results which are often used by subsequent executions.
- Iterative operations, such as generalization and suppression algorithm, in SparkDA are cached then re-used which results in high-performance.
- Our experimental results illustrate the feasibility of our proposal by showing that data utility is still high when compared with its counterpart in standalone operation (i.e., data utility is not destroyed as the results of taking advantage of high-performance).
- Results show the scalability of SparkDA while maintaining high-performance.

The rest of this paper is organized as follows: Sect. 2 provides the background that is related to this paper. Section 3 explains the details of our proposed method SparkDA and two data anonymization RDD transformation algorithms. Section 4 discusses our experimental setup, results and key findings in details. Section 5 provides the conclusion along with future work.

2 Background

This section provides the background materials to better understand our proposal. The section starts with the main architectural ideas behind MapReduce and Spark, their similarities and differences. An illustration of the main idea behind a data anonymization technique based on Datafly [14] is presented.

2.1 MapReduce vs Spark

The Hadoop MapReduce [9] has been a popular big data processing platform for the last decade. The MapReduce programming paradigm is based on Map and Reduce. MapReduce starts with multiple mappers on various nodes based on data locality to process the mappers in parallel at each node. The input data, typically large, is split into mappers of multiple nodes. The data in a mapper, which is a collection of records either structured or unstructured, is assigned with key-value combinations. The mapper writes key-value pairs in the

local disk at each data node. The reduce function is designed to collect mappers' results. The reducer reads these key-value pairs from local disk and exchanges the relevant keys to the respective reducers. Figure 1 illustrates the execution cycle of a MapReduce job to highlight data movement from an input to an output. However, some performance degradation in the current MapReduce paradigm can occur in the following places.

- Problem (1): When splitting the input data, the size and the number of splits decide the number of mappers at each node – often with no knowledge of the capability of mappers. One such allocation is done, there is no movement of mappers across nodes. This creates several performance issues. For example, it can create a long execution queue if a node contains too many mappers as MapReduce only use the memory of each node for processing the mapper at the local node. Any subsequent mappers have to wait until the memory is freed even though other nodes could be idle. Subsequently, it also creates delay in the reducer as it needs to wait until this busy node with many mappers finishes all processing even though the mappers in other nodes have already finished much earlier. This problem is illustrated as "Problem 1" in Fig. 1.
- Problem (2): Mappers create intermediate values which are written in the local disks at each node in MapReduce. This creates a several trips to the expensive disk I/O as the number of intermediate results increases. This problem is illustrated as "Problem 2" in Fig. 1.
- Problem (3): The reducer produces the final results. All intermediate results across multiple nodes require to be transferred to the reducer which could often locate in another node in the network. The transmissions of intermediate results across multiple nodes and the reducer can also create performance degrade. This problem is illustrated as "Problem 3" in Fig. 1.

These problems results in a tremendous performance bottleneck with iterative operations. In the current MapReduce architecture, a severe execution queue is created with iterative operations which require a series of sequential MapReduce jobs, where each iteration to be waited and executed one by one until all iterations are done.

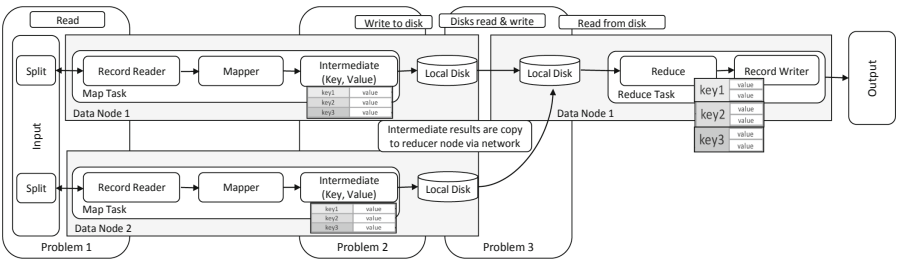


Fig. 1. Components and data flow in the MapReduce execution cycle

To address the performance concern of MapReduce, Spark has emerged as a high-performance distributed processing platform for big data. Spark uses Resilient Distributed Datasets (RDDs), which are immutable collection of records partitioned in a parallel manner. Input data is read from the disk as a split block as it was done in MapReduce environment, however, the blocks are further split into several partitions. An input RDD is created which contains all the partitions. The input RDD understands the memory capability at each worker's node, and by taking this account, assigns the optimal number of partitions to each node. This can effectively reduce the issue we discussed in the earlier problem (1). Once partitions are done, more RDDs are created to transform the original data. The intermediate values at each RDD transformation are written in the memory rather than the disk. This architecture can effectively remove the performance degradation mentioned in the problem (2).

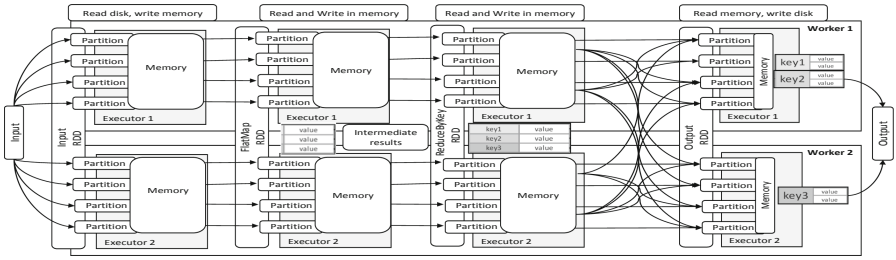


Fig. 2. Components and data flow in Spark execution cycle

In MapReduce, each node is executed as a separate unit where the intermediate values are not shared but being hold at its respective node. The intermediate results across different nodes occur at the reducer as a result of transferring data. This is not necessary in the Spark model as each RDD has the global knowledge of the previous stages and their intermediate results. We illustrate the execution flow of Spark in Fig. 2 which reads the input data in the memory, pre-process it, and then transform it through RDDs.

2.2 Data Anonymization

Data anonymization involves transforming an original data into an anonymized data in such a way where individually identifiable attributes or tuples in the original data are changed to a set of indistinguishable attributes or tuples. The transformation typically utilizes two techniques; generalization and suppression, respectively.

- Generalization refers to a process where the value of an attribute is replaced with a less specific value. In general, generalization is based on a Domain Generalization Hierarchy (DGH) associated to that attribute. A DGH specify a Generalization Level (GL) for each attribute. DGH is usually provided by a domain expert based on the attribute characteristics.

- Suppression replaces the attribute with the one that do not release any information about the attribute at all.

Figure 3 illustrates the general generalization approach that applies generalization levels (GLs) defined in a DGH. $GL5$ in "Date of Birth" represents an example of suppression with "*". Among many variations of data anonymization exists, our approach follows the basic idea from Datafly [14].

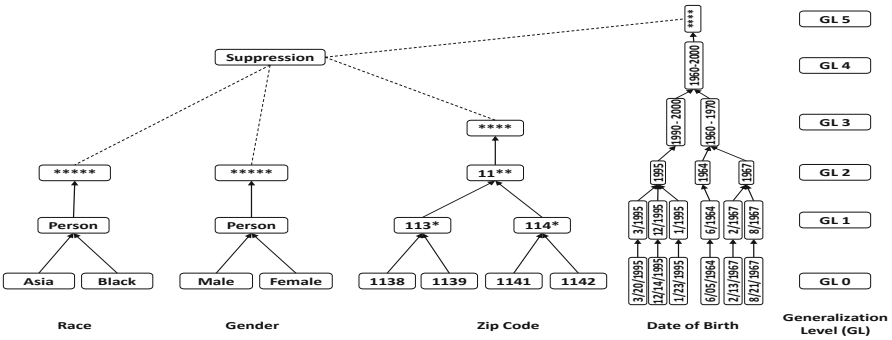


Fig. 3. Examples of Generalization and Domain Generalization Hierarchies including suppression

The data anonymization in this approach starts by counting the frequency over the Quasi Identifiers Attributes (QID). Then, it generalizes the attribute having the most distinct values until k-anonymity [15] is satisfied. The QID refers to a set of attributes that can be used to uniquely identify an individual (e.g. data of birth, age, and address).

Table 1 illustrates how the original data, depicted in Table 1(a), can be anonymized. To start the transformation, the algorithm first computes the frequency of each tuple and the distinct values of each attributes. These frequency counts is written as illustrated in Table 1(b). In the next step, the algorithm uses a DGH (such as one seen in Fig. 3) to perform a generalization step to satisfy k-anonymity rules from the attribute having the most number of distinct values. For example, "Date of Birth" is first generalized as it has the most number of distinct attributes values followed by "Zip Code". One or multiple levels of generalizations are applied from the generalization hierarchy until it satisfies k-anonymity – see Table 1(c).

Table 1(d) shows the effect of the full generalization/suppression with respect to the frequency list update. For example, for $k = 2$ anonymity, "Date of Birth" is generalized to level = 5, "Zip Code" to level = 3, followed by "Gender" and "Race" up to level = 2. Table 1(d) represents $k = 2$ anonymization as final result.

Table 1. Data anonymization steps

(a) Original Data

Date of Birth	Gender	Zip Code	Race	Disease
3/20/1995	Female	2141	Asian	Fever
12/14/1995	Female	2141	Asian	Back Pain
1/23/1995	Male	2138	Asian	Chest Pain
6/05/1964	Female	2139	Black	Brocken Hand
2/13/1967	Male	2138	Black	Asthma
8/21/1967	Male	2138	Black	Heart Attack

(b) Frequency counts

Date of Birth	Gender	Zip Code	Race	Frequency	Tuple
3/20/1995	Female	2141	Asian	1	T1
12/14/1995	Female	2141	Asian	1	T2
1/23/1995	Male	2138	Asian	1	T3
6/05/1964	Female	2139	Black	1	T4
2/13/1967	Male	2138	Black	1	T5
8/21/1967	Male	2138	Black	1	T6
6	**2**	**3**	**2**		

(c) Partially Anonymized Data

Date of Birth	Gender	Zip Code	Race	Frequency	Tuple
1995	Female	2141	Asian	2	T1, T2
1995	Male	2138	Asian	1	T3
1964	Female	2139	Black	1	T4
1967	Male	2138	Black	2	T5,T6
3	**2**	**3**	**2**		

(d) Fully Anonymized Data

Date of Birth	Gender	Zip Code	Race	Disease
1995	Female	2141	Asian	Fever
1995	Male	2138	Asian	Back Pain
1964	Female	2139	Black	Asthma
1967	Male	2138	Black	Heart Attack

3 SparkDA

We first outline the basic symbols and notations in Table 2 to clearly define the elements of data across different scopes in a dataset. Figure 4 illustrates a

Table 2. Basic symbol and notations

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

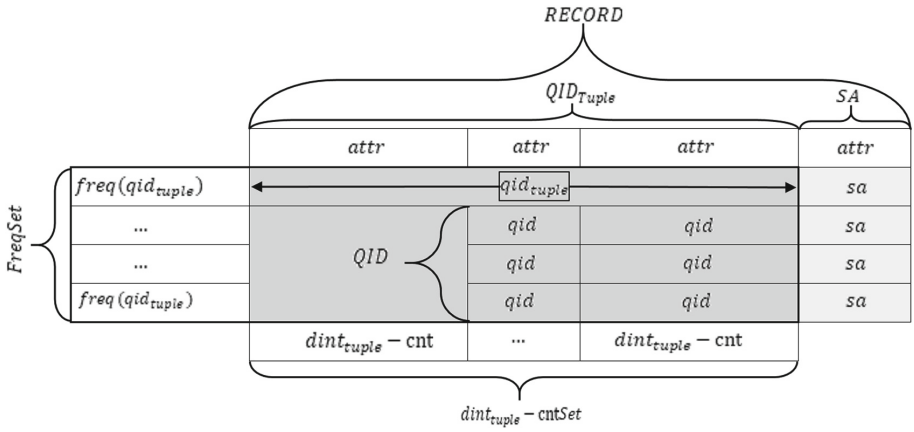


Fig. 4. Notations representation for any given table

diagram as how our notations can be mapped into a relational database table. In the followed sections, we describe the details of two main RDD transformations where the main idea of our anonymization techniques is applied – FlatMapRDD and RedueByKeyRDD followed by the description of our proposed approach SparkDA.

3.1 RDD-Based Data Anonymization

We have implemented new data anonymization supports in two Spark RDD transformations, FlatMapRDD and ReduceByKeyRDD, respectively.

FlatMap Transformation (FlatMapRDD): The FlatMapRDD runs an algorithm to get the frequency counts for distinct tuples that contains all quasi-identifiable attributes as well as for the distinct variations within each quasi-identifiable attribute. The details of the FlatMapRDD algorithm are shown in Algorithm 1.

The start of the FlatMapRDD algorithm, it requires QID_{Tuple} as an input. In the initial stage, the QID_{Tuple} contains the original quasi-identifiable attributes. The first part of the algorithm, from step 2–11, is used to get the frequency count for distinct set of quasi-identifiable attributes. This is done by first measuring the size of QID_{Tuple} to identify the total number of qid_{tuple} it contains (in step 3). Then, the current qid_{tuple} is compared with the subsequent qid_{tuple}. If there is a match between qid_{tuple}(s), a frequency count is updated by adding the number 1. This is done for each qid_{tuple} within the QID_{Tuple}. By step 10, $FreqSet$ for all qid_{tuple}(s) is updated with the frequency counts for each unique tuple.

Algorithm 1. FlatMapRDD

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

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

ReduceByKey Transformation (ReduceByKeyRDD): The main purpose of ReduceByKeyRDD is to run an iteration of (anonymization) transformation from the given $FreqSet$ and $dint_{qid}$-$cntSet$. Here, the transformation refers to the change such as taking place in Table 1(a) and (b), and from Table 1(b) and (c), and so on until reaches Table 1(d). It utilizes an "anonymization statue ($anonymization_s$)" to identify whether a given QID_{Tuple} is fully anonymized or if it further needs to run more transformations. Algorithm 2 describes the ReduceByKeyRDD.

The algorithm starts by receiving the (DGH, K) which was sent via a broadcast mechanism from Spark driver node, which runs the main SparkDA algorithms. (DGH, K), as notation implies, contains both the Domain Generalization Hierarchies (DGH) and the size of K-group (K). Once received, DGH is further processed to extract the generalization level (GL) for each quasi-identifiable attribute as seen in step 3 and 4.

Algorithm 2. ReduceByKeyRDD

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

The first part of the algorithm, the steps 6–18, is to generalize attributes up to a single generalization level for all quasi-identifiable attribute sets. This is done if the frequency counts (freq(qid_{tuple})) has not exceed the size of K (k-anonymization) and while the maximum generalization level (MAX(GL_{qid})) has not met. The single generalization level is done in the order of the attributes with the highest distinct attribute counts (MAX($dint_{qid}$-cntt)) to lower. The anonymization status is set to false as there is more transformation to be done.

The second part of the algorithm, the steps 20–26, is to suppress all attributes for a given tuple that have violated k-anonymity rules – that is, there exist no indistinguishable tuples. At this stage, all transformation is done including the suppression. The anonymization status is set to true as there is no more transformation to be done. The anonymized results (either or both being

generalized/suppressed) are sent back to FlatMapRDD along with the anonymization status (as seen in step 29). FlatMapRDD will re-calculate the frequency counts for tuples and attributes when anonymization status is false.

3.2 Overall SparkDA Scheme

This part describes the overall process of our SparkDA that are associated not only with the data anonymization but also other parts that assist the data anonymization process.

To run the SparkDA, it first needs an input dataset (i.e., the original data) along with other user defined information such as the size of K and the definition for DGH. The user defined information is used as global variables that can be shard across all Spark worker nodes that process RDDs (such as ReduceByKeyRDD). The global variables can be sent via the use of broadcast mechanism in Spark.

Algorithm 3 illustrates the overall pseudo-code. The algorithm starts by reading the file from HDFS as an input dataset which are subsequently stored by the InputRDD. The InputRDD processes the input data in a way that is easier for other RDDs to progress. For example, it splits the input data into two separate sets, one set containing all quasi-identifiable attributes (QID_{Tuple}-RDD) while the other set containing all sensitive attributes (SA-RDD) – seen in step 5.

Algorithm 3. SparkDA

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

```
1  begin
2      InputRDD ← textFile(Dataset)
3      broadcast(DGH, K) ← broadcast(DGH)
4      broadcast(DGH, K) ← broadcast(K)
5      anonymization_s = false
6      SA-RDD, QID_Tuple-RDD ← InputRDD.filter(qid_tuple, sa)
7      SA-RDD_c ← SA-RDD.cache
8      QID_Tuple_c ← QID_Tuple-RDD.cache
9      while anonymization_s = false do
10         Result-RDD(QID_Tuple, anonymization_s) ←
           QID_Tuple.FlatMapRDD(QID_Tuple)
           .ReduceByKeyRDD(dint_qid-cntSet, FreqSet)
11         QID_Tuple-RDD.cache ← filter.Result-RDD(QID_Tuple,
           anonymization_s)
12         QID_Tuple_c ← QID_Tuple-RDD.cache
13         anonymization_s ← filter.Result-RDD(QID_Tuple, anonymization_s)
14     end
15     Anonymized_Tuple ← filter.Result-RDD(QID_Tuple, anonymization_s)
16     Anonymized(RDD) ← Anonymized_Tuple.join(SA-RDD_c)
17     return Anonymized(RDD)
18 end
```

Both $SA\text{-}RDD$ and $QID_{Tuple}\text{-}RDD$ are cached in memory for further processing. The cached QID_{Tuple_c} is used by FlatMapRDD and ReduceByKeyRDD for data anonymization as described in the above section. The anonymization status at this stage is set to false to execute FlatMapRDD and ReduceByKeyRDD to signal the start of the anonymization process. If anonymization is finished, which fully QID_{Tuple} is returned from ReduceByKeyRDD, then the anonymization status is set to true. At this stage, QID_{Tuple} only contains the distinct qid_{tuple} but are not joined with the sensitive value. The joining between QID_{Tuple} and $SA\text{-}RDD$ only happens in step 19.

4 Experimental Results

In this section, we first explain our experimental setups that include the details of the dataset and the system environment (hardware/software) configurations. This is followed by the description of data utility metrics we used and the results we obtained to understand the information loss as the results of our data anonymization. In addition, we also provide the results of scalability and performance.

4.1 Datasets

We used two datasets in our study: US Census dataset (often described as Adult dataset) [4] and Irish Census dataset [1]. We downloaded the original Adult dataset, then synthesized it to create a set of larger datasets for the experiments. Similarly, we downloaded already synthesized Irish dataset and further created more data from it. The synthesized datasets are generated by using Java open-source tool "Benerator" [5]. We followed the guideline from [6] to increase the number of records. Table 3 illustrates the quasi-identifiable attributes (QID) we used in our experiments, and generalization level (GL) of each QID obtained from the domain generalization hierarchy (DGH) for both the datasets. The "Salary" in Adult dataset and the "Field of Study" in Irish dataset are set as sensitive attributes.

Table 3. Datasets

(a) Adult dataset

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

(b) Irish dataset

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

4.2 System Environment Configurations

We ran two types of experiments, first one with distributed processing platform using Spark and the other with standalone desktop. The standalone desktop environment was used to understand the comparability of data utility results – this should stay the same though our data anonymization technique takes advantage of scalability and high-performance features of Spark. For Spark, as a distributed processing platform, we configured Yarn and Hadoop Distributed File System (HDFS) using Apache Ambari. HDFS distributes data in a NameNode (worked as a master node), a secondary NameNode and six DataNodes (worked as worker nodes). We configured 3 GB memory for Yarn NodeManager while 1 GB memory was allocated to ResourceManager, Driver, and Executor memories each. We used Spark version 2.1 along with Yarn as a cluster manager. Table 4(a) illustrate the Spark and Hadoop Parameters. Table 4(b) depicts the details of the spark cluster and standalone computer and their respective CPU, Memory, Disk, and Network speed (Gbit/s). Note that we used a Windows 10 as a standalone desktop. We ran our experiments 10 times and the average was used to ensure the reliability and consistency of the results.

Table 4. Hardware and cluster configuration

(a) Spark and Hadoop Parameters

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

(b) Hardware Configuration

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

4.3 Privacy and Utility Trade-Offs

We used the following four privacy and utility metrics to validate and understand the rate of information loss between the original data and the anonymized datasets produced as the results of running our SparkDA algorithms.

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 result in the higher data utility as it is more difficult to identify an attribute among many identical attributes. In k-anonymized dataset, the size of the equivalence classes is greater than or equal to K. As a result, the quality of the data is lower if the size of all or part of the equivalence classes greatly exceeds the value K. The score of C_{AVG} sensitive to the K group size [11]. C_{AVG} for AnonymizedRDD is calculated as following.

$$C_{AVG} = \frac{\mid AnonymizedRDD \mid}{\mid EC \mid} / K \tag{1}$$

$| \mathit{AnonymizedRDD} |$ represents the total number of records of *Anonymized RDD*, whereas $| EC |$ represents the total number of equivalence classes.

Precision Metric (*PM*): *PM* [14] is used to choose the least distorted records (i.e., both from attributes and tuple perspective) from the set containing all anonymized records. *PM* is sensitive to the *GL*. Following equation defines PM_{score} for AnonymizedRDD.

$$PM_{score} = 1 - \frac{\sum_{qid=1}^{qid_{tuple}} \sum_{qid_{tuple}=1}^{QID_{Tuple}} \frac{GL}{|DGH_{qid_{tuple}}|}}{qid_{tuple}.QID_{Tuple}} \tag{2}$$

Where *GL* defines a generalization level (defined in *DGH*) including suppression. Attributes with higher generalization level typically maintains a rate of precision better when compared to attributes with lower generalization levels.

Table 5. Experimental configurations for data utility

#	Utility metrics	Anonymization parameters	Dataset size	Platform
1	C_{AVG}, *PM*	*K-value* $\in$ {2, 5, 10, 25, 50, 75, 100}, $\mid QID \mid = 5$[a]	Adult = 30K	Spark, standalone
		K-value $\in$ {2, 5, 10, 25, 50, 75, 100}, $\mid QID \mid = 5$[a]	Irish = 30K	Spark, standalone

[a]Indicates the total number of attributes, we use 5 attributes in the experiment

To understand the privacy and utility trade-offs, we varied the K group sizes in our experiment. This implies that the increased in the K group size would obviously make data utility reduced - which would results in the increase in data privacy. What it means is that as there are more data made indistinguishable, it is harder to identify an individual. Note that $K2$ depicts that two tuples to become indistinguishable while $K5$ to have five tuples to be same so on. The experimental setup for privacy vs. utility is shown in Table 5. We ran two experiments; one on Spark platform and the other on the standalone desktop against two different datasets.

The results of data utility metrics for SparkDA and standalone are illustrated, first with the Adult dataset Fig. 5(a)–(b) then for Irish dataset Fig. 5(c)–(d).

We first discuss the data utility results of Adult dataset. C_{AVG} measure data utility based on the equivalence class. The data utility decreases with the increase in the size of K group as there are more distinct attributes for matching. As there are more data matching between two equivalent classes, it implies the data privacy is high but data utility is low. However, the average penalty remains the same because at some point, approximately around $K = 10$, data is no longer either generalized or suppressed in an equivalence class hence no changes in the penalty value. Thus, the average penalty of an equivalent class decreases as the number of K group size increases which is seen in Fig. 5(a).

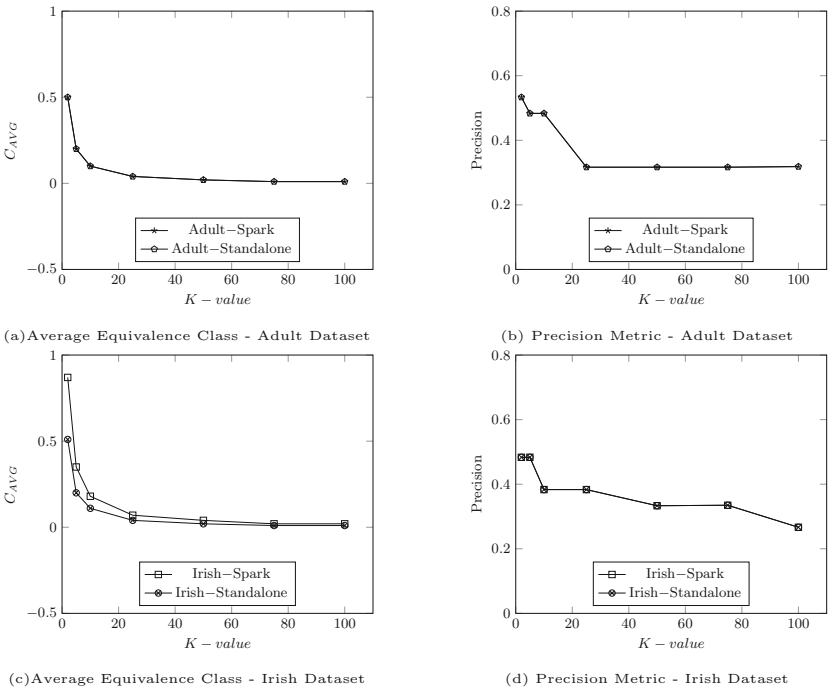


Fig. 5. Data utility vs Adult dataset and Irish dataset on Spark and standalone

Precision Metric (PM), in Fig. 5(b), demonstrates the level of distortion at the record level (i.e., the combination of tuples and attributes). It is expected that PM score will be higher as the number of K group size increases as there are more records that have lost its original values. The PM score is highly sensitive to GL for each qid. This is shown in Fig. 5(d) where the PM score increases as the number of K group size increases for both Spark and standalone. This is because the level of GL applied in each qid is increased to its highest as the size of K group increases. We observe that at $K = 25$ and onward, the qid are appeared to have been generalized to its highest level as the PM score stays the same.

The data privacy and utility results for Irish dataset shows slightly different data utility scores. We observe that in overall, Irish dataset contains the records that are more distinct from each other. The data utility increases with the increase in the size of K group where there are more distinct quasi-identifiable attributes; this is shown in C_{AVG} score in Fig. 5(c). The PM scores in Fig. 5(d), are same for Spark and standalone environment ensuring that the data privacy and utility were not affected between two implementations.

4.4 Scalability and Performance

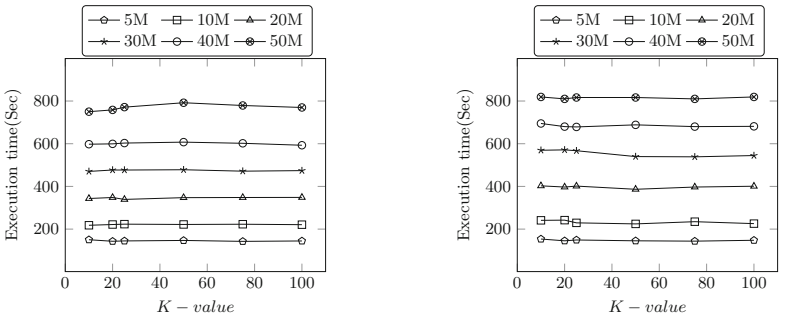
We perform experiments to understand scalability and performance of our proposal. We used the increasing size of records and measure the execution for different K group size. The execution time includes both FlatMapRDD and ReduceByKeyRDD transformation. The details of the experiments are illustrated in Table 6.

Table 6. Experimental configurations for scalability and performance

#	Experiment	Anonymization parameters	Dataset size
1	Records size	K-*value* $\in$ {10, 20, 25, 50, 75, 100}, $\mid QID \mid = 5$[a]	Adult = (5M,10M, 20M, 30M, 40M, 50M)
			Irish = (5M,10M, 20M, 30M, 40M, 50M)

[a]It indicates the number of attributes that were used in the experiments

The experiments are aimed at understanding the impact of the number of records on various K-group size in Fig. 6(a) and (b).



(a) Adult dataset (b) Irish dataset

Fig. 6. Efficiency of SparkDA for varied record size

We first only increased the size of K-group on a fixed number of records to understand the relationship between the execution time and the size of K-group. Results show that the execution time appears not to be affected by increasing K group size. The number of iterations from the original data to fully anonymized dataset is decided based on the frequency of distinct tuples. The number of K group size would increase the number of tuples. With the fixed number of QIDs, the number of tuples that are increased doesn't necessarily are distinct. This means the frequency count stays the same. With the frequency count remaining the same, the same number of operations are done irrespective to the increasing number of K-size thus the execution time stays the same.

This appears that some operations (e.g., involved in QID generalization) are cached in memory then re-used and this does not affect too much on execution time. However, this changes as soon as the number of records is increased. The execution time linearly increases as the number of records increase Fig. 6(a) and (b) in both datasets.

5 Conclusion and Future Work

We proposed a novel data anonymization approach name SparkDA for Spark platform. Our data anonymization algorithms are implemented in FlatMapRDD and ReduceByKeyRDD to take the full advantages of many innovations in Spark which includes; better partition control, in-memory processing, and cache management. Our experimental results show high data utility scores compared to a standalone version meaning that the level of privacy is preserved while taking advantage of Spark features. Our experimental results also illustrate the linear grows of the execution time in line with the increasing number of QID(s) and the number of records. The linear grows, without any visible peaks and anomalies, validates that our proposed approach is feasible by supporting scalability with performance.

In future, we plan to extend our study in the number of areas. First, we plan to validate the cache performance on different storage levels and its effectiveness. Secondly, we plan to better understand Spark's support for shuffle and sort operations and their impacts. We also plan to add a number of privacy metrics. Furthermore, we also plan to extend our current study to work with multidimensional attributes.

References

1. Central Statistics Office (Internet) (2011). http://www.cso.ie/en/databases/. Accessed 16 Aug 2019
2. Al-Zobbi, M., Shahrestani, S., Ruan, C.: Sensitivity-based anonymization of big data. In: IEEE 41st Conference on Local Computer Networks Workshops (LCN Workshops), pp. 58–64. IEEE (2016)
3. Antonatos, S., Braghin, S., Holohan, N., Gkoufas, Y., Mac Aonghusa, P.: Prima: an end-to-end framework for privacy at scale. In: IEEE 34th International Conference on Data Engineering (ICDE), pp. 1531–1542. IEEE (2018)
4. Asuncion, A., Newman, D.: UCI machine learning repository (2007). http://archive.ics.uci.edu/ml. Accessed 16 July 2019
5. Ayala-Rivera, V., McDonagh, P., Cerqueus, T., Murphy, L.: Synthetic data generation using benerator tool. arXiv preprint arXiv:1311.3312 (2013)
6. Ayala-Rivera, V., McDonagh, P., Cerqueus, T., Murphy, L., et al.: A systematic comparison and evaluation of k-anonymization algorithms for practitioners. Trans. Data Priv. **7**(3), 337–370 (2014)
7. Bazai, S.U., Jang-Jaccard, J., Wang, R.: Anonymizing k-NN classification on MapReduce. In: Hu, J., Khalil, I., Tari, Z., Wen, S. (eds.) MONAMI 2017. LNICST, vol. 235, pp. 364–377. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-90775-8_29

8. Bazai, S.U., Jang-Jaccard, J., Zhang, X.: A privacy preserving platform for MapReduce. In: Batten, L., Kim, D.S., Zhang, X., Li, G. (eds.) ATIS 2017. CCIS, vol. 719, pp. 88–99. Springer, Singapore (2017). https://doi.org/10.1007/978-981-10-5421-1_8
9. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
10. Grolinger, K., Hayes, M., Higashino, W.A., L'Heureux, A., Allison, D.S., Capretz, M.A.: Challenges for MapReduce in big data. In: IEEE World Congress on Services, pp. 182–189. IEEE (2014)
11. LeFevre, K., DeWitt, D.J., Ramakrishnan, R., et al.: Mondrian multidimensional k-anonymity. In: ICDE, vol. 6, p. 25 (2006)
12. Shi, J., et al.: Clash of the titans: MapReduce vs. spark for large scale data analytics. Proc. VLDB Endow. **8**(13), 2110–2121 (2015)
13. Sopaoglu, U., Abul, O.: A top-down k-anonymization implementation for apache Spark. In: IEEE International Conference on Big Data (Big Data), pp. 4513–4521. IEEE (2017)
14. Sweeney, L.: Achieving k-anonymity privacy protection using generalization and suppression. Int. J. Uncertain. Fuzziness Knowl. Based Syst. **10**(05), 571–588 (2002)
15. Sweeney, L.: k-anonymity: a model for protecting privacy. Int. J. Uncertain. Fuzziness Knowl. Based Syst. **10**(05), 557–570 (2002)
16. Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.: Spark: cluster computing with working sets. HotCloud **10**(10–10), 95 (2010)
17. 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: 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, pp. 501–508. IEEE (2013)