electronics

MDPI

Article

In-Memory Data Anonymization Using Scalable and High Performance RDD Design

Sibghat Ullah Bazai * and Julian Jang-Jaccard

Cybersecurity Lab, Computer Science/Information Technology, Massey University, Auckland 0632, New Zealand; j.jang-jaccard@massey.ac.nz

* Correspondence: s.bazai@massey.ac.nz

Received: 18 August 2020; Accepted: 1 October 2020; Published: 20 October 2020

Abstract: Recent studies in data anonymization techniques have primarily focused on MapReduce. However, these existing MapReduce based approaches often suffer from many performance overheads due to their inappropriate use of data allocation, expensive disk I/O access and network transfer, and no support for iterative tasks. We propose "SparkDA" which is a new novel anonymization technique that is designed to take the full advantage of Spark platform to generate privacy-preserving anonymized dataset in the most efficient way possible. Our proposal offers a better partition control, in-memory operation and cache management for iterative operations that are heavily utilised for data anonymization processing. Our proposal is based on Spark's Resilient Distributed Dataset (RDD) with two critical operations of RDD, such as FlatMapRDD and ReduceByKeyRDD, respectively. The experimental results demonstrate that our proposal outperforms the existing approaches in terms of performance and scalability while maintaining high data privacy and utility levels. This illustrates that our proposal is capable to be used in a wider big data applications that demands privacy.

Keywords: high performance; data anonymization; scalability; spark; big data mining; privacy and utility

1. Introduction

The rapid growth of data from many domains (e.g., social media, smartphones, IoT etc.) has brought in a new era where extracting potential information using data analytic and data mining has become a top business priority to many organizations. Such practices, however, have also brought up data privacy concern in the absence of appropriate data protection mechanisms.

Data anonymization approaches are used to conceal private information in such a way where identifiable (sensitive) information is buried among non-identifiable groups [1,2]. Many different data anonymization algorithms have been proposed for the purpose including *K*-anonymization [3], *l*-diversity [4], *t*-closeness [5] and others [6,7].

The recent growth of big data has created a high demand for distributed processing platforms that are equipped with a core set of features, for example, scalable processing units, large execution engines, and high capacity storage. Many existing anonymization methods used to run on a single machine have been redesigned to work with these new platforms (e.g., MapReduce) as the size of the input data increases massively [8–10].

In addition, many existing researches show that data anonymization methods implemented on MapReduce platform often have performance bottlenecks because underlying platform does not have appropriate supports for many core anonymizations tasks. These includes; MapReduce does not have a support for allocating data across partitions in different nodes in a balanced fashion which increases network overhead, doesn't support cache operation for saving the data produced while a task is still

processing (e.g., intermediate data) which results in the intermediate data often being stored in the disk and fetched whenever it is needed [11], and does not have a support for iterative tasks which increases many performance overheads in terms of memory and network management [12]. More details of the issues associated with MapReduce are discussed in Section 3.1.

To address the limitations of MapReduce, Reference [13] proposed a new platform named Spark and since been hailed as the next generation of distributed processing platform. Spark has extended its scalability aspect in addition to offering a new set of advanced features more suited for the algorithms dealing with many different types of big data operations [14]. With the surge in the population of Spark and shift from MapReduce approach, many Spark-based data anonymization techniques have been proposed [15–19]. However, these existing proposals often tend to focus their efforts on improving and readdressing the scalability aspects to be more suited for Spark instead of investigating the suitability of Spark as a platform of choice for data anonymization techniques.

This is an extension of the earlier version which we presented in Reference [20]. The focus of the original paper was to present the details of a novel data anonymization approach based on Spark to take the full use of the advanced features offered by Spark while this extension offers an extensive evaluation for the suitability of our proposal for data anonymization techniques. By adapting and improving the advanced features of Spark, our approach effectively addresses many shortcomings of existing MapReduce based data anonymization approaches to resolve the overheads associated with expensive disk I/O, network and iteration tasks. We have extended our earlier version in several aspects. The new contributions of this paper are listed as follows:

- We provide clearer example of a general approach involved in a basic data anonymization technique with the addition of a flowchart to assist the understanding of the main tasks involved in such a technique. In addition, an additional mapping table is provided to further illustrate the relationship between the symbols and notations we use and database concept.
- We provide more detailed description of two critical RDDs involved in our proposal, FlatMapRDD and ReduceKeyByRDD respectively. These are designed to provide a better partition management, in-memory access for various data produced during anonymization process, and an effective cache management. We provide a better description as how our RDD-based approach can effectively reduce the significant overhead associated with MapReduce counterparts.
- We provide a new performance comparison between our proposal and the most up to date existing K anonymity based approaches and evaluates that our proposal offers a very competitive performance advantage.
- In addition to additional utility measurement matrices for Discenibility Metric (DM) and Minimal Distortion (MD), we provide a new set of privacy measurement matrices, such as Kullback-Leibler-Divergence (KLD) and Information Entropy (I_E), to extensively investigate the privacy and utility trade-offs of our proposal.
- We also provide the insights of a new set of performances associated with different memory management strategies offered by Spark. We discover that side-effect can occur when there are too excessive demands for memory access.

The paper is structured as follows. In Section 2, we provide the recent related studies, while in Section 3, we provide the issues associated with MapReduce approach along with the description of a basic data anonymization technique as backgrounds. In Section 4, we discuss the details of our proposal along with main algorithms involved in our RDDs. Section 5 describes the details of the number of privacy and utility matrices we utilise and how we use them in the context of our proposal. In Section 6, we discuss the results of our experiments and the key findings. Section 7 provides the conclusion and planned future work.

2. Related Work

To address the overheads associated with MapReduce, a number of Spark based approaches have been proposed in recent years. Reference [21] proposed the INCOGNITO framework for full-domain generalization using Spark RDDs. Though their experiential results illustrate the improvement in scalability and execution efficiency, their proposal does not provide any insights of privacy and utility trade-offs. Anonylitics [19] utilized Spark's default iteration support to implement data anonymization. However, their approach does not address the potential memory exhaustion unable to accommodate increasing number of intermediate data produced as the number of iteration increases. PRIMA [16] proposes an anonymization strategy for Mondrian algorithm with Optimal Lattice Anonymization (OLA) which is used to define the utility and generalization level rules in order to limit the data utility loss. Reference [22] proposes a distributed Mondrian approach by splitting the input data to the partitions allocated to each node of cluster by using Spark *k*-mean. A series of Spark jobs runs on each cluster node to produce anonymized results. These anonymized results are then merged together later by another cluster node.

The study that is most close to ours is that in Reference [17] which provided a distributed Top-Down Specialization (TDS) algorithm to provide *K*-anonymity using Apache Spark. Rather, their solution focuses on addressing scalability and partition management which was originally proposed by Reference [23]. They neither provide the details of the Spark feature they utilized nor any insights of privacy and utility trade-offs. Al-Zobbi et al. [18] proposed a sensitivity-based anonymization using user-defined function in Spark. The authors provide a strategy for reduced data transmission between memory and disk based on serialized data objects implemented with RDD and validate that a Spark-based approach can be many times faster than MapReduce counterparts such as Reference [11].

3. Background

We first provide the comparison of the difference and issues involved in MapReduce and Spark. This is followed by the description of the main tasks involved in a basic data anonymization strategy (e.g., Datafly [24]).

3.1. MapReduce vs. Spark

For many years until Spark, Hadoop MapReduce [8] has been a widely used distributed processing platform for many big data applications. The fundamental building blocks of MapReduce are Map and Reduce. At start, MapReduce divides the (large) input data into a several smaller chucks. Each chunk of data (i.e., typically a collection of records) is mapped to a map across multiple mappers. The data contained in a mapper is assigned for a key-value combination. Each mapper process the data based on the key-value pair and the results, often called as intermediate data, is stored in the local disk where the mapper resides. Once the processing of all mappers are complete, a reducer reads the results from all mappers. Figure 1a shows the full execution cycle of a MapReduce job and data movements involved at each phase. We argue that many performance overheads occur while MapReduce executes a job, especially in the following phases.

- Problem 1: One of the implications associated with MapReduce is with the creation of mappers where the size and number of mappers are decided without the consideration of the capability of each node. Once data is allocated to mappers, it is not possible to re-allocate records across different mappers. This creates a several performance issues. Consider a case where a mapper is allocated with a larger set of records compare to other mappers. The execution for this mapper requires the use of the majority of the memory at the local disk while other mappers attached to the same node which shares the same local disk have to wait until the memory is freed. This can cause creating a long execution queue. Consequently, it can also cause a massive delay in the reducer in which waits for a long time until the mapper with the larger dataset completes despite all other

mappers have already completed and their results are available much earlier. This problem is demonstrated as "Problem 1" in Figure 1.

- Problem 2: Each mapper writes the results of the processing at the mapper in the local disk as intermediate data. The reducer requires accessing the intermediate data for further processing. This can cause the increase of expensive disk I/O by the mappers and the reducer when the number of intermediate data increases. This problem is demonstrated as "Problem 2" in Figure 1.
- Problem 3: In MapReduce, a reducer processing the results of many mappers may reside in a separate network node. In this case, the results of mappers (i.e., intermediate data) requires to first read from the disk associated with the mapper, transferred across the network, and finally saved in the reducer's disk. As number of mappers increase, this can cause a significant network bottleneck especially if a particular network is slow or unavailable. This problem is demonstrated as "Problem 3" in Figure 1.
- Problem 4: In case of a task with iterative nature, the result is first written in the local disk. If this result needs to be used again in the subsequent iteration, the mapper needs to access the disk again for each iteration. This architectural design is not only ineffective but also results in a tremendous performance bottleneck as it would cause a severe execution queue. To avoid the queue, the developer of MapReduce requires creating a series of sequential MapReduce jobs for the mappers manually. Even with this choice, it is often necessary that each iteration is waited for the completion (due to the issue discussed in the Problem 1).

Spark utilises Resilient Distributed Datasets (RDDs) as the building block to process Spark jobs. RDDs hold immutable collection of records which are partitioned and can be processed separately in parallel. Similar to MapReduce, input data is spilt as several smaller blocks. Each block then can be further divided into several partitions. An input RDD is created to hold all the partitions in the beginning. It then assigns partitions in the manner accounting for the processing capability at each worker's node to have the optimal number of partitions that can work most effective at each node. This new capability of Spark can reduce the issue associated with the Problem 1 we discussed earlier.

Once the initial partition allocation is complete, more RDDs are created to process the data contained in each partition – this is called a transformation in Spark. The intermediate data created by each RDD transformation is written in the memory and referenced as necessary. The memory accessibility can effectively reduce the performance overhead we discussed in the Problem 2 and 4.

In MapReduce jobs, the execution of each node happens as a separate unit of work. The result of each node, the collection of intermediate data, is not shared but being written off at each node due to the data locality principle of MapReduce. The only way to share the intermediate data with a reducer is via data transfer across networks. Spark offers the data sharing across different RDDs including the results produced by the previous stages and the intermediate data produced by different RDDs. This new feature of Spark can address the concerns we discussed in the Problem 3 and 4.

The execution flow of Spark is illustrated in Figure 1b from data reads off the input data to the memory, processing data at different partitions, and then processing the partitions through RDD transformations.

3.2. Data Anonymization

Data anonymization refers to a process of transforming a set of original data into an anonymized data in such a way that uniquely identifiable attributes no longer present in the anonymized dataset while preserving statistical information about the original dataset. Two separate techniques are used for data transformation: generalization and suppression, respectively.

- Generalization involves with a process to replace the value of an attribute to a less specific value. Domain Generalization Hierarchy (*DGH*), which is typically defined and provided by a domain expert, is used to find the granularity for the generalization levels to be applied for each attribute.
- Similar to generalisation, suppression involves with a process replacing the original attribute to the value that does not release any statistical information about the attribute at all.

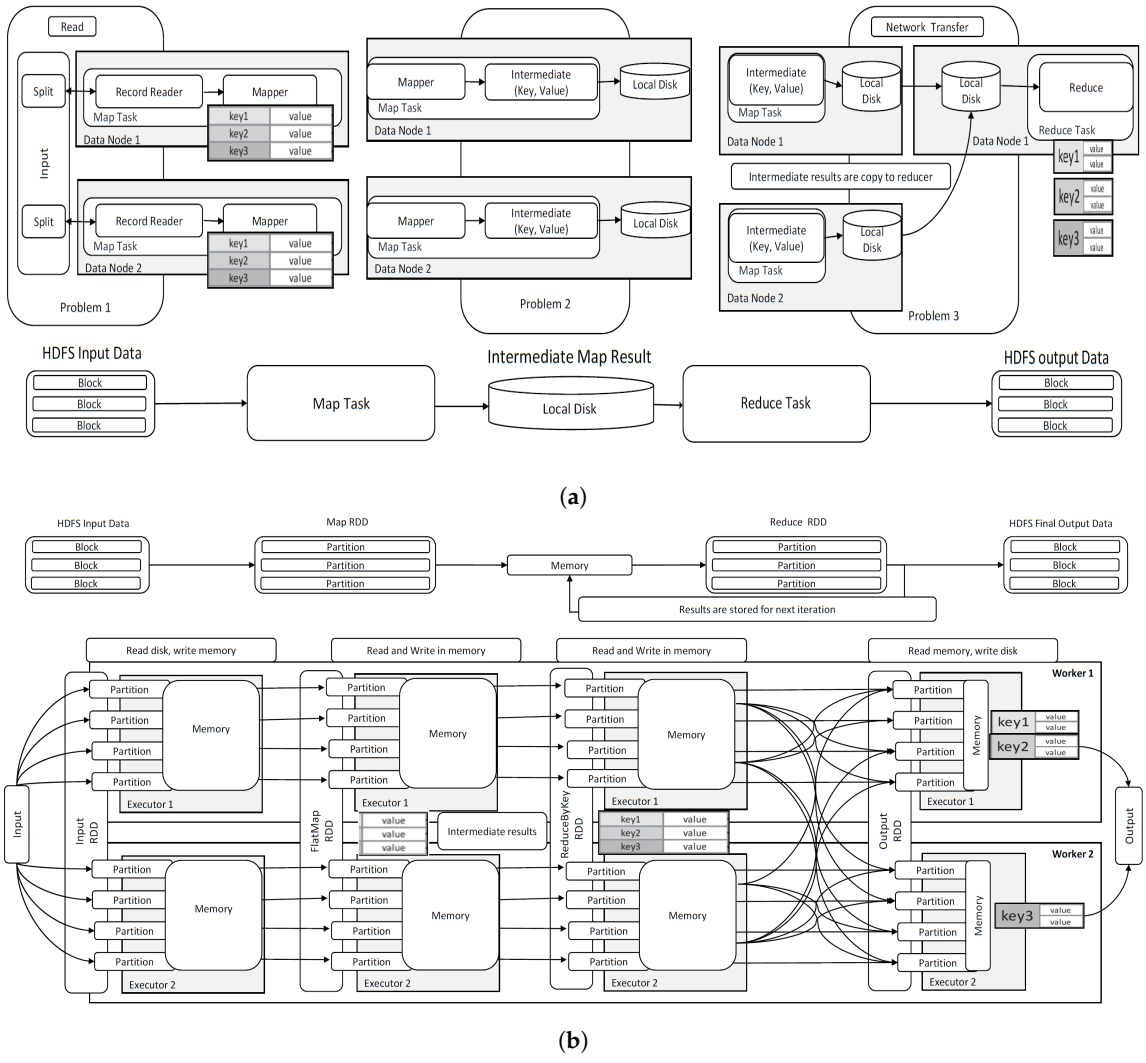

Figure 1. Comparing the components and Data-flow in MapReduce and Spark structures. (**a**) MapReduce structures; (**b**) Spark structures.

Figure 2 demonstrates a generalization approach for applying generalization levels (GLs) defined in a *DGH*. For example, GL0 represents the first level of generalization while higher levels of generalizations are presented by GL2 and GL3. "*" is an example of suppression which appears in many attributes as the highest generalization level. Each "*" represents a numerical value of a generalization level, such as 114* represents GL1 while 11**, 1*** and * represents GL2, GL3, and GL4 respectively.

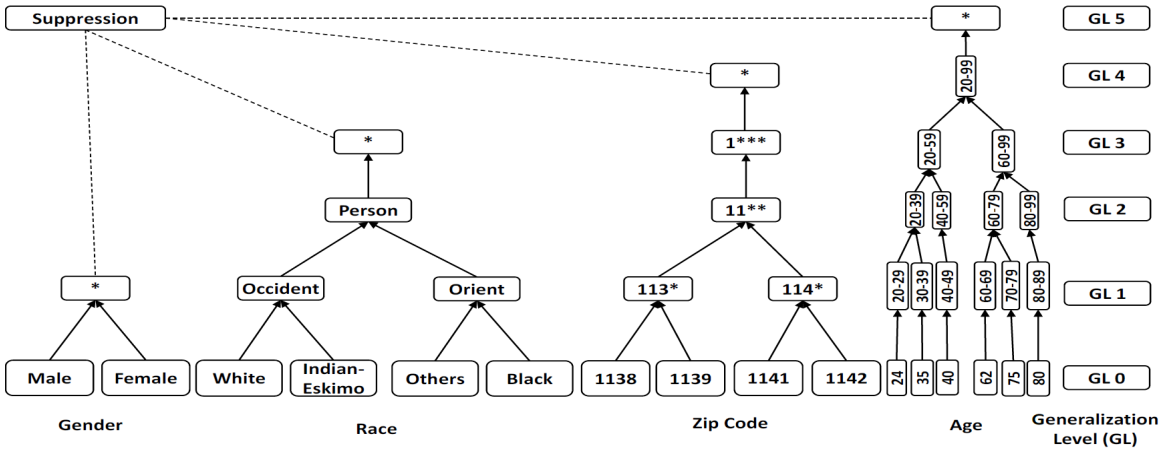

Figure 2. Examples of Generalization and Suppression for a Domain Generalization Hierarchies (*DGH*).

Though many variations of data anonymization methods have been proposed, our approach follows the one that is similar to Datafly [24]. The flow of Datafly algorithm is depicted in Figure 3. In this approach, data anonymization starts by counting the frequency, which represents the number of appearances given the record set, over the Quasi Identifiers Attributes (*QID*). The *QID* refers to a set of attributes that can uniquely distinguish an individual (e.g., age, date of birth, or address). Taking from the attribute with the most number of frequency count, the technique generalizes each attribute until *K*-anonymity constraint [3] is fully satisfied.

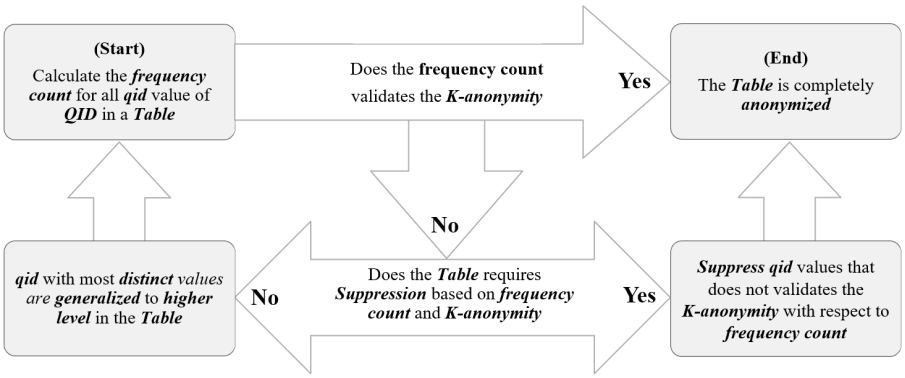

Figure 3. Datafly Algorithm

Table 1 illustrates the number of iterations in which a generalization is applied from the original data to a fully anonymized dataset. It starts with the original data depicted in Table 1 (a). The original data is transformed based on the counting of the frequency of unique attributes and the frequency of unique tuples. Table 1 (b) now contains the frequency counts. Starting from the attribute with the highest number of the frequency count, generalization based on *DGH*, an example shown in Figure 2, is applied. For example, the attribute "Age" is first generalized because it has the highest number of the frequency count at 6. Table 1 (c) depicts a partially anonymized data. Note that a multiple level of generalizations can be performed at this stage as long as it doesn't violate the *K*-anonymity constraint. The final fully anonymized result is presented in Table 1 (d) which meets the $K = 2$ constraint.

Table 1. Data Anonymization Steps.

(a) Original Data					
Age	Gender	Zip Code	Race	Disease	
24	Female	1141	White	Fever	
35	Female	1141	White	Fever	
40	Male	1138	White	Back Pain	
62	Female	1139	Black	Asthma	
75	Male	1138	Black	Heart Attack	
85	Male	1138	Black	Heart Attack	
(b) Frequency Counts					
Age	Gender	Zip Code	Race	Frequency	Tuple
24	Female	1141	White	1	T1
35	Female	1141	White	1	T2
40	Male	1138	White	1	T3
62	Female	1139	Black	1	T4
75	Male	1138	Black	1	T5
85	Male	1138	Black	1	T6
6	**2**	**3**	**2**		
(c) Partially Anonymized Data					
Age	Gender	Zip Code	Race	Frequency	Tuple
20–59	Female	1141	White	2	T1,T2
20–59	Male	1138	White	1	T3
60–99	Female	1139	Black	1	T4
60–99	Male	1138	Black	2	T5,T6
3	**2**	**3**	**2**		
(d) Fully Anonymized Data					
Age	Gender	Zip Code	Race	Disease	
20–59	Female	1141	White	Fever	
20–59	Male	1138	White	Back Pain	
60–99	Female	1139	Black	Asthma	
60–99	Male	1138	Black	Heart Attack	

4. SparkDA

In this section, we describe the details of our approach named SparkDA. We first provide the descriptions for the symbols and notations we used. Then, we describe our two RDDs, FlatMapRDD and RedueByKeyRDD, and the algorithms each of the RDDs executes.

4.1. Basic Symbols and Notations

The elements of the data across different scopes are outlined using the symbols and notations in Table 2. The mapping diagram of our proposed notations to a relational database concept is demonstrated in Figure 4.

Table 2. Basic Symbols and Notations.

Symbol	Definition
PT	A table (dataset) that contains records
RECORD(r)	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}, $dint_{qid}$-$cntSet = \{dint_{qid}$-$cnt_1, \ldots, dint_{qid}$-$cnt_{attr}\}$
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

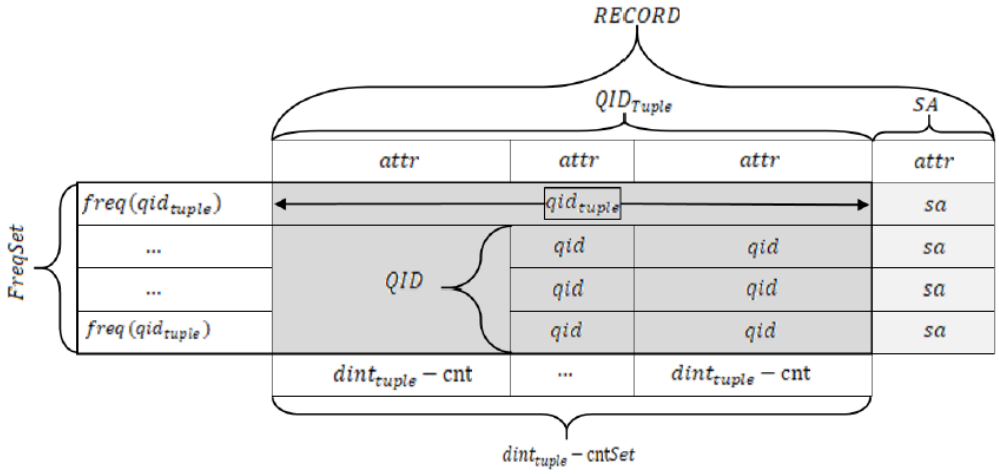

Figure 4. Notations Mapped for a Database Table.

4.2. RDD-Based Data Anonymization

In our proposed approach, a data anonymization technique is implemented through the use of two Spark RDD transformations, FlatMapRDD and ReduceByKeyRDD, respectively.

4.2.1. FlatMap Transformation (FlatMapRDD)

The overall purpose of the FlatMapRDD is to compute for both the frequency of distinct attributes and the distinct tuples for all quasi-identifiable attributes. The frequency counts are then used to decide if further anonymization is necessary.

The Algorithm 1 illustrates the working of the FlatMapRDD algorithm. The algorithm starts by loading the input data into QID_{Tuple}. At this initial stage, the QID_{Tuple} contains the original quasi-identifiable attributes.

Algorithm 1: FlatMapRDD.

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

begin
  $freq(qid_{tuple}) = 1$
  for *i in Size*(QID_{Tuple}) **do**
    if $qid_{tuple_i} = qid_{tuple_{i+1}}$ **then**
      $freq(qid_{tuple}) + +$
    end
    $FreqSet+ = (qid_{tuple}, freq(qid_{tuple}))$
  end
  $dint_{qid}$-$cnt = 0$
  for *i in Size*(QID_{Tuple}) **do**
    for *j in Size*(qid_{tuple}) **do**
      $QID_j = qid_{tuple(i)(j)}$
    end
  end
  for *i in Size*(QID) **do**
    if $qid_i = qid_{(i+1)}$ **then**
      $dint_{qid}$-$cnt_{(i)}$
    end
    else
      $dint_{qid}$-$cnt_{(i)} + +$
    end
    $dint_{qid}$-$cntSet$ += $dint_{qid}$-$cnt_{(i)}$
  end
  return ($FreqSet$, $dint_{qid}$-$cntSet$)
end

The first part of the algorithm (depicted by step 2–8) executes to identify the frequency counts. To do this, it first measures the size of QID_{Tuple} to compute the total number of qid_{tuple} it contains (in step 3). The current qid_{tuple} is compared to the next qid_{tuple}. If a match is found between the two comparing qid_{tuple}(s), the frequency count is updated by adding the number 1. This is repeated for each and every qid_{tuple} within the QID_{Tuple}. However, the algorithm does not update frequency count if the qid_{tuple} and the subsequent qid_{tuple} values are different as this indicates two different records. When the iteration through QID_{Tuple} completes, the frequency counts for each unique tuple for all qid_{tuple}(s) is saved in the *FreqSet* (seen in step 7). It should note that Spark sorts the qid_{tuple}(s) within the partition of each executing node and the frequency count of each qid_{tuple} is always equal to the number of respective qid_{tuple} appearing in the dataset as the total frequency count for all qid_{tuple}(s) represent the sum of records in the dataset.

The second part of the algorithm (depicted by step 9–22) runs to identify the count for the distinct attribute within a QID. To do this, it first measures the size of QID_{Tuple} to compute the total number of QID(s) it contains. Subsequently, the current qid is compared to the next qid. If a match is found between the two qid(s), the distinct qid count is updated by adding the number 1. This is repeated for each and every qid given the QID. When the iteration through QID(s) completes, the distinct counts for each unique attribute for all qid(s) is saved in the $dint_{qid}$-cnt (seen in step 22). The algorithm returns *FreqSet* and $dint_{qid}$-$cntSet$ along with QID_{Tuple} to ReduceByKeyRDD.

4.2.2. ReduceByKey Transformation (ReduceByKeyRDD)

The overall aim of the ReduceByKeyRDD is to execute an RDD transformation by applying a generalization level using the information contained in *FreqSet* and $dint_{qid}$-$cntSet$. The RDD transformation can be interpreted as the changes made to the original data in Table 1 (a) until it reaches the results seen in Table 1 (d), through Table 1 (b) and Table 1 (c). We introduce an "anonymization statue (represented by a variable = $anonymization_s$)" to keep track of whether a given QID_{Tuple}, which contains the lasted anonymization results, is fully anonymized or not and if a further anonymization processing is necessary. The Algorithm 2 illustrates the working of the ReduceByKeyRDD algorithm. To start the algorithm, the combination of (DGH, K) which contains the taxonomy tree and the K-anonymity constraint, is received via a broadcast mechanism which is sent by the driver node. DGH is further used to retrieve the generalization level (GL) for each quasi-identifiable attribute. This is described in step 3–4.

The first part of the algorithm (depicted by steps 6–18) is operated to apply a single generalization level in all quasi-identifiable attribute sets. Applying a generalization level is repeated until the frequency counts (freq(qid_{tuple})) does not exceed the size of K and also does not exceed the maximum generalization level (MAX(GL_{qid})). The generalization is applied to attributes with the highest distinct attribute counts (MAX($dint_{qid}$-cnt)) to lower. The anonymization status is set to false while generalization level is being applied.

The second part of the algorithm (depicted by steps 21–26) is operated by applying suppression for all attributes for a given tuple which have violated the K-anonymity constraint to ensure no indistinguishable tuples exists. By now, all anonymization is complete, including the suppression, therefore the anonymization status is set to true. As seen in step 29, the anonymized results are sent back to the FlatMapRDD along with the anonymization status. Upon receiving updated QID_{Tuple} which now contains the anonymized data, the FlatMapRDD computes again for the frequency counts for the distinct tuples and the distinct attributes if only the anonymization status is still set to false.

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
```

4.3. Overall SparkDA Scheme

In this section, we describe the overall process of our proposed approach that includes both the data anonymization process by two RDDs we described earlier and how these RDDs interact with other parts of the program.

The overall algorithm for our SparkDA is illustrated in Algorithm 3. The algorithm runs first by reading off user defined information such as K (i.e., K-anonymity constraint) and DGH (i.e., contains the definition of generalization hierarchy), as depicted in step 3–4. The K and DGH are used as global variables that are shard across all Spark worker nodes associated with processing RDDs. Spark supports broadcast mechanism to send the global variables across worker nodes.

The original data file from HDFS is read and saved into an InputRDD (step 1). The InputRDD pre-processes the input data in such a way that is easier to be processed by other RDDs. For example, the input data is divided into two different datasets, one set contains all quasi-identifiable attributes (QID_{Tuple}-RDD) while the other set contains all sensitive attributes (SA-RDD) (step 6). We cache SA-RDD and QID_{Tuple}-RDD as they are used in many subsequent processing. At this stage, the anonymization status is set to false (step 5).

As depicted in steps 9–14, now two RDDs involved in data anonymization process, FlatMapRDD and ReduceByKeyRDD, executes interactively many times. The anonymization process completes when the fully anonymized dataset QID_{Tuple} is returned from ReduceByKeyRDD in which the anonymization status is set to true. The anonymized dataset, a generalized and distinct qid_{tuple} contained within QID_{Tuple}, is finally joined with corresponding SA-RDD (step 16).

Algorithm 3: SparkDA.

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

1 **begin**
2 $InputRDD \leftarrow textFile(Dataset)$
3 $broadcast(DGH, K) \leftarrow broadcast(DGH)$
4 $broadcast(DGH, K) \leftarrow broadcast(K)$
5 $anonymization_s = false$
6 $SA\text{-}RDD, QID_{Tuple}\text{-}RDD \leftarrow InputRDD.filter(qid_{tuple}, sa)$
7 $SA\text{-}RDD_c \leftarrow SA\text{-}RDD.cache$
8 $QID_{Tuple_c} \leftarrow QID_{Tuple}\text{-}RDD.cache$
9 **while** $anonymization_s = false$ **do**
10 $Result\text{-}RDD(QID_{Tuple}, anonymization_s) \leftarrow QID_{Tuple}$.FlatMapRDD($QID_{Tuple}$) .ReduceByKeyRDD($dint_{qid}$-$cntSet$, $FreqSet$)
11 $QID_{Tuple}\text{-}RDD.cache \leftarrow filter.Result\text{-}RDD(QID_{Tuple}, anonymization_s)$
12 $QID_{Tuple_c} \leftarrow QID_{Tuple}\text{-}RDD.cache$
13 $anonymization_s \leftarrow filter.Result\text{-}RDD(QID_{Tuple}, anonymization_s)$
14 **end**
15 $Anonymized_{Tuple} \leftarrow filter.Result\text{-}RDD(QID_{Tuple}, anonymization_s)$
16 $Anonymized(RDD) \leftarrow Anonymized_{Tuple}.join(SA\text{-}RDD_c)$
17 return $Anonymized(RDD)$
18 **end**

The details of Spark execution cycle according to the overall SparkDA operations is depicted in Figure 5.

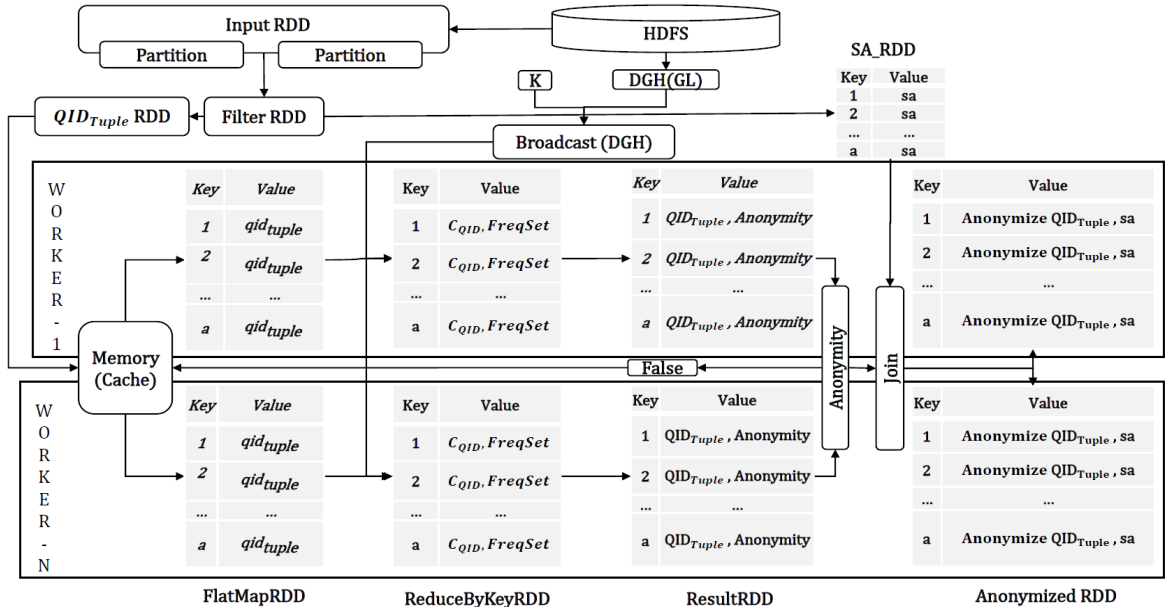

Figure 5. DataFlow in Spark.

5. Privacy vs. Utility Trade-Offs

We used the following privacy and utility metrics to validate and understand the trade-offs between these two. In the study of understanding the success of a data anonymization technique, a privacy level is measured by identifying the uniqueness of data. With that, a low privacy typically means that it is easy to identify an individual (an attribute, tuple or record) from a group (e.g., many records are unique) while a high privacy indicates that it is (more) difficult to uniquely identify an individual from a group (e.g., there are many records sharing the same values). A utility level is measured by calculating the level of degradation in accuracy of value between the original value (i.e., baseline) and the anonymized value (i.e., sanitized).

5.1. Privacy Metrics

5.1.1. Kullback-Leibler−Divergence (*KLD*)

KLD is utilized for understanding the likelihood of the presence of the original attribute in the anonymized attribute for each record [25]. For example, assume that the original attribute of the age 24 is anonymized into a range of 20–59. The *KLD* can measure what is the possibility of guessing the original age 24 from the range 20–59. Note that we use the term "likelihood" instead of "probability" to indicate that our calculation is done on the past event of the known outcomes (i.e., anonymized dataset). We measure *KLD* on the fully anonymized dataset by computing the followings: (1) calculating the likelihood of the presence of each attribute, (2) sums up all the value of (1) for each attribute within a record, then continues steps (1) and (2) for all records. Here, $P_{InputRDD}$ indicates the sum of the likelihood of the presence of the original attribute within the original data (at a record level). $P_{InputRDD}$ at this stage has a very high data utility and no privacy as there is no changes made. $P_{InputRDD(r)}$ indicates the sum of the likelihood of the presence of the original attribute within the anonymized record. $P_{AnonymizedRDD}$ usually has lost some degree of data utility and has gained some degree of privacy because the data in this set has changed from the baseline after an anonymization technique is applied.

$$KLD = \sum_{r=1}^{n} P_{AnonymizedRDD}(r) \log \frac{P_{AnonymizedRDD}(r)}{P_{InputRDD}(r)}. \quad (1)$$

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).

5.1.2. Information Entropy (I_E)

The I_E is used to measure the degree of how uncertain it is to identify the original value from the anonymized value within the QID attributes [26]. The entropy value of I_E is 1 if all the qid attributes are identical in the anonymized dataset for the same QID. The $I_E(QID)$ value can be calculated by, (1) calculating the likelihood of the presence of the original attribute in a record, (2) computing the sum of the values of step (1) for each attribute in a record (denoted as $P_{AnonymizedRDD(qid)}$), (3) continuing the steps (1) and (2) for each QID, and (4) computing the sum of the value of step for all records. Note that if all attributes are changed between the original record and the anonymized record, the value of $P_{AnonymizedRDD}$ is 1.

$$I_E = -\sum_{qid=1}^{n} P_{AnonymizedRDD(qid)} \log P_{AnonymizedRDD(qid)}. \tag{2}$$

From Equation (2), we obtain $I_E(QID)$ for a single QID, however, we are interested in the I_E for the whole anonymizedRDD. Thus, we calculate the I_E for anonymizedRDD by taking the average of all QIDs. The entropy value of I_E is 0 if there are two identical records from the original dataset to the anonymized dataset for a matching equivalent class. The maximum value of I_E is achieved when the original record sets is completely different from the anonymized record sets for a given QID. Higher value of I_E represents more uncertainty (i.e., higher privacy).

5.2. Utility Metrics

5.2.1. Discernibility Metric (DM)

DM reports the data quality resulting from the degree of data degradation, as a result of data anonymization, of an individual tuple based on an equivalent class. Let EC be the set of equivalence classes of a K-anonymized dataset. EC_i is one of the equivalence classes of $\mid EC \mid$. The DM metric can be expressed more formally for AnonymizedRDD as follows:

$$DM_{score} = \sum_{EC_i \in AnonymizedRDD} \mid EC_i \mid^2, \tag{3}$$

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.

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

C_{AVG} measures data utility of attributes by calculating the average size of the equivalence class. A higher data utility is typically achieved when the number of equivalence size is bigger because it is more difficult to distinguish an attribute when there are large number of attributes. Therefore, it is considered that the results of C_{AVG} scores are sensitive to the K group size [27]. We calculate C_{AVG} according to AnonymizedRDD as following.

$$C_{AVG} = \frac{|AnonymizedRDD|}{|EC|} / K, \tag{4}$$

where $|AnonymizedRDD|$ denotes the total number of records within the anonymized set while the total number of equivalence classes is denoted by $|EC|$.

5.2.3. Minimal Distortion (MD)

The MD measures data utility of every quasi-identifiable attribute (qid) in a tuple (qid_{tuple}). It defines data utility by comparing the rate where how many numbers of qid(s) in (qid_{tuple}) have been made to be indistinguishable. This is done by measuring the level of distortion on each qid in respect to a generalized level [28]. We calculate the distortion from the qid_{tuple} of AnonymizedRDD in comparison to InputRDD by using the following equation.

$$MD = \sum_{i=1}^{|D|} MD[Inputqid_{tuple-i}, Anonymizedqid_{tuple-i}], \tag{5}$$

where $|D|$ depicts the number of tuples in InputRDD. Equation (5) defines MD for complete dataset. The overall distortions between the anonymized dataset and the original dataset can be minimized by decreasing the K group size.

5.2.4. Precision Metric (PM)

As cited in Reference [24], PM calculates the least distorted combination of attribute and tuples from anonymized records. PM is typically considered to be sensitive to the GL. We define the equation for PM_{score} according to AnonymizedRDD as follows.

$$PM_{score} = 1 - \left(\frac{\sum_{qid=1}^{qid_{tuple}} \sum_{qid_{tuple=1}}^{QID_{Tuple}} \frac{GL}{|DGH_{qid_{tuple}}|}}{qid_{tuple}.QID_{Tuple}} \right), \tag{6}$$

where GL represents a generalization level (including suppression) which is defined in the DGH. The attributes associated with a higher generalization level tends to provide a better precision score than the attributes with a lower generalization level.

6. Experimental Results

This section first illustrates our experimental setups with the dataset and the system environment configurations. Then, we discuss the results of privacy and utility scores we obtained. The comprehensive experimental results of scalability, performance, and the impact of different cache management strategies of Spark follows.

6.1. Datasets

In our study, we used two datasets: US Census dataset (i.e., Adult dataset) [29] and Irish Census dataset [30]. We synthesized these datasets to increase the number of records to investigate different aspects of performance. We used "Benerator", which is a Java-based open-source tool, and the guideline from Reference [31] to generate the synthesized datasets. Table 3 illustrates the details of the both datasets including the quasi-identifiable attributes (QID), the number of district value, and generalization levels. The sensitive attributes are set to the "Salary" in the Adult dataset and the "Field of Study" in the Irish dataset.

6.2. System Environment Configurations

Our experiments were run on two different platforms. The first sets of experiment were executed in a distributed processing platform environment using Spark while the other sets of experiment were executed on a standalone desktop. The latter was used to validate the comparability of data privacy and utility. The expectation was that the data privacy and utility scores should stay same between two sets of experiments.

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

We used Spark 2.1 where Yarn and Hadoop Distributed File System (HDFS) were configured using Apache Ambari. HDFS was used to distribute data across a NameNode (worked as a master node), a secondary NameNode, and six DataNodes (worked as worker nodes). 3 GB memory was allocated to Yarn NodeManager while 1 GB memory was configured for each of ResourceManager, Driver, and Executor memory. Table 4 (a) shows the Spark and Hadoop Parameters while Table 4 (b) provides the details of the Spark cluster and standalone desktop setups. Windows 10 was used as a standalone desktop. All experiments ran at least 10 times and the average was used as to warrant the reliability and consistency of the results.

Table 4. Hardware and Cluster Parameters and Configuration.

(a) Spark and Hadoop Parameters			
Spark		Hadoop	
Resource Manager Memory	1 GB	NameNode	1
Driver Memory	1 GB	DataNode	6
Executor Memory	1 GB	Block Replication	3
Driver Cores	1	Block Size	128 MB
Executor Cores	1	HDFS Disk	18 TB

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

6.3. Privacy and Utility

We discuss the results of running privacy and utility metrics in this section. We illustrated the details of experimental in Table 5.

Table 5. Experimental Configurations for Data Privacy and Utility.

#	Metrics	Anonymization Parameters	Dataset Size	Platform
1	DM, C_{AVG}, MD, PM	$K\text{-}value \in \{2, 5, 10, 25, 50, 75, 100\}$, $\mid QID \mid = 5$ *	Adult = 30 K	Spark, Standalone
		$K\text{-}value \in \{2, 5, 10, 25, 50, 75, 100\}$, $\mid QID \mid = 5$ *	Irish = 30 K	Spark, Standalone
2	KLD, I_E	$K\text{-}value \in \{2, 5, 10, 25, 50, 75, 100\}$, $\mid QID \mid = 5$ *	Adult = 30 K	Spark, Standalone
		$K\text{-}value \in \{2, 5, 10, 25, 50, 75, 100\}$, $\mid QID \mid = 5$ *	Irish = 30 K	Spark, Standalone

* Indicates the total number of attributes. Here there are 5 attributes in the experiment.

6.3.1. Privacy Results

The results of *KLD* metric on Adult dataset are shown in Figure 6a. The results show that the *KLD* values stay identical between Spark and standalone environment which means the implementation of data anonymization in Spark didn't affect any privacy level. 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 results of *KLD* metric on Irish dataset are shown in Figure 6b. In general, the overall observation of the changes of *KLD* values is similar to that of Adult dataset. However, we observe that the average *KLD* values are much higher in the Irish dataset than Adult dataset. This is due that the Irish dataset has more generalization levels for each *QID* which increase the chances of more number of *QID*s to share the same value. This increases a privacy level.

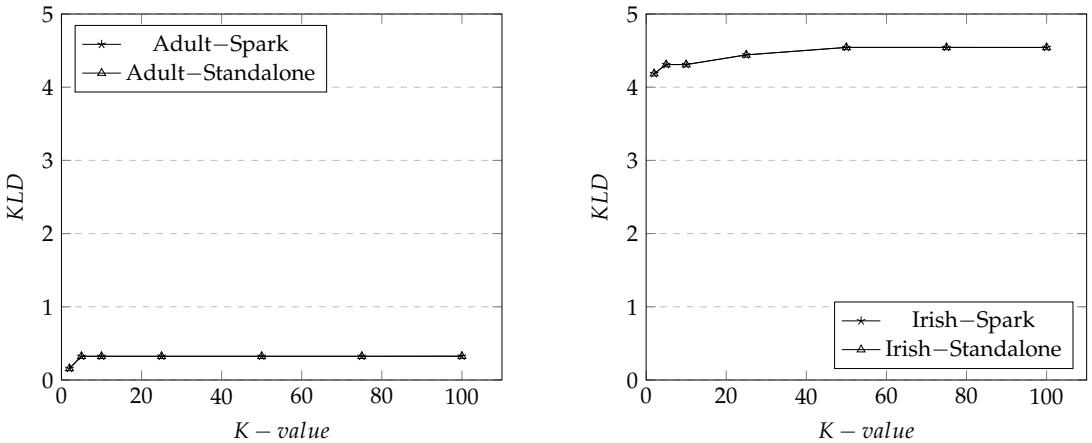

Figure 6. Divergence for Adult and Irish datasets on both Spark and Standalone. (**a**) KL -Divergence in Adult Dataset; (**b**) KL-Divergence in Irish Dataset.

The results of I_E metric on Adult dataset are shown in Figure 7a. Again, the values between the Spark and Standalone remain the same which ensures that the implementation of our data anonymization technique in Spark didn't destroy the privacy level. The average of I_E values in Adult dataset is lower compare to Irish dataset.

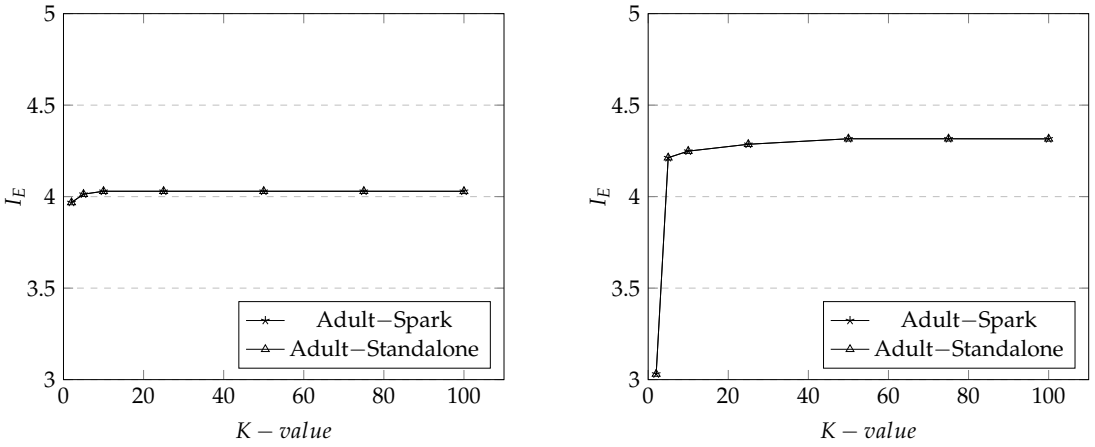

Figure 7. Information Entropy for Adult and Irish datasets on both Spark and Standalone. (**a**) I_E in Adult Dataset; (**b**) I_E in Irish Dataset

Our investigation reveals that Adult dataset contains relatively the small number of different QIDs which share the same value as the result of anonymization. The smaller K value affects the I_E value more compare to the greater K value due to the number of same values in QID attributes. This affects in the higher I_E value as it is easier to identify a unique record within the same equivalent class compare to Irish dataset which has a larger number of different QIDs that share the same value.

6.3.2. Utility Results

We illustrate the results of data utility metrics, based on the results obtained from Adult dataset Figure 8a,c,e,g and from Irish dataset Figure 8b,d,f,h.

We first discuss the data utility results of Adult dataset. The overall DM scores produced by both Spark and standalone are relatively high at 0.9. Recall that DM measures the data utility of tuples within an equivalent class. It is expected that the increased in the K group size would result in the increase in the equivalent class. As the equivalent class becomes larger, there will be more changes to make tuples to be more indistinguishable which would result in a high DM score—the results represented in Figure 8a. In addition, there is a sudden increase in the DM score approximately around K = 5 both in the Spark and standalone. This illustrates that at K = 5 and onwards the degradation of data has reached the maximum and there is no more generalization/suppression to be applied (i.e., data utility is at the lowest).

The trend of C_{AVG} scores were similar to DM as both metrics were based on the calculation according to the size of equivalence classes. We observe the trend where the data utility scores decline when the size of K group increases as there are more matched distinct attributes. The average penalty seem to remain same at around K = 10 with no changes in generalization. The rationale is that at this point, there are no more generalizations or suppressions to apply to an equivalence class. As a consequence, the average penalty for an equivalent class drops when the number of K group size grows. This is seen in Figure 8b.

Figure 8c illustrates the results of *MD* which measures the rate of data utility based on the changes made to tuples from the original dataset to the anonymized dataset. It is expected that *MD* score would increase when the *K* group size increases because there would be more attributes in a tuple not matching between the original dataset and the anonymized dataset. *MD* tends to be more sensitive to generalization levels because the attributes in a tuple applied with higher generalization levels would have more dramatic changes.

Precision Metric (*PM*), in Figure 8d, demonstrates the level of distortion at the record level (i.e., the combination of tuples and attributes). It is expected that the *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 Figure 8d 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.

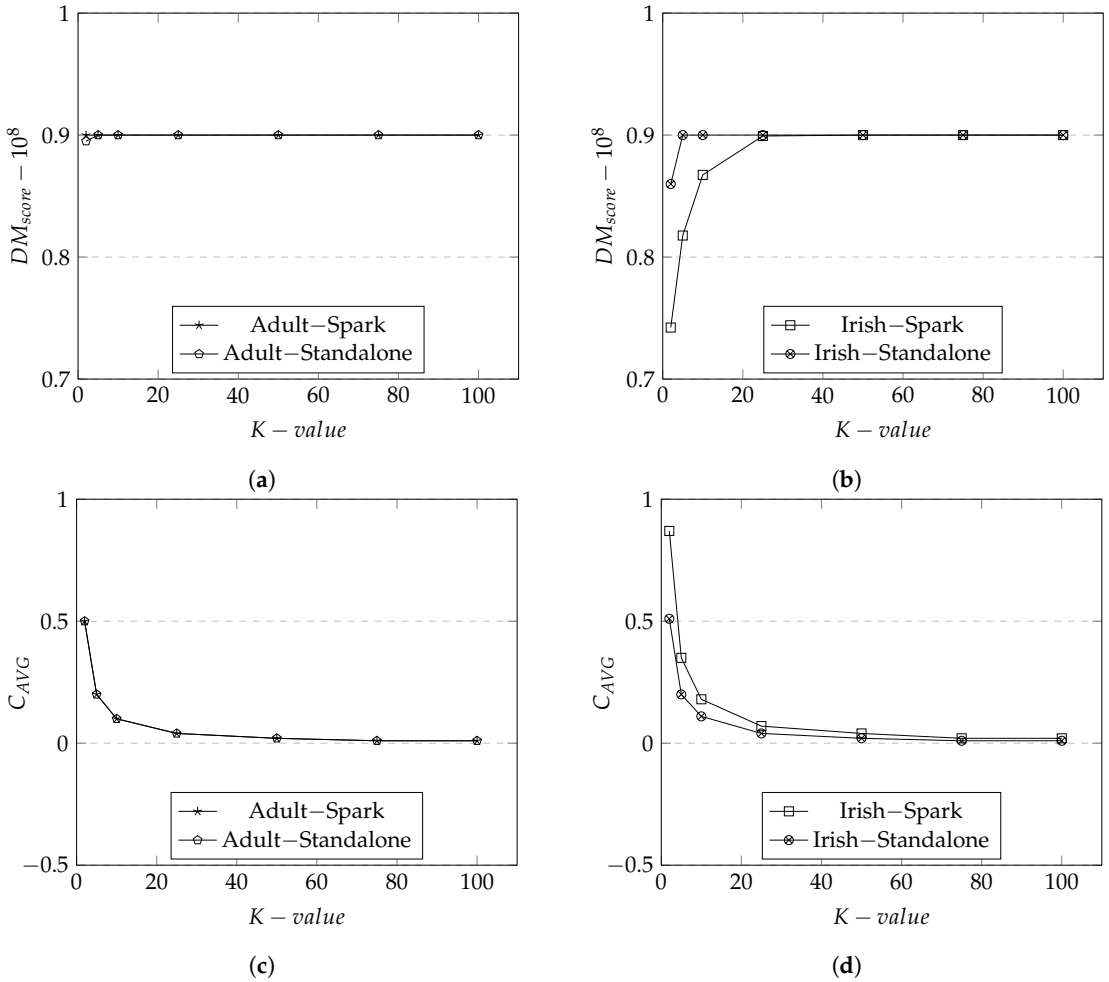

Figure 8. *Cont.*

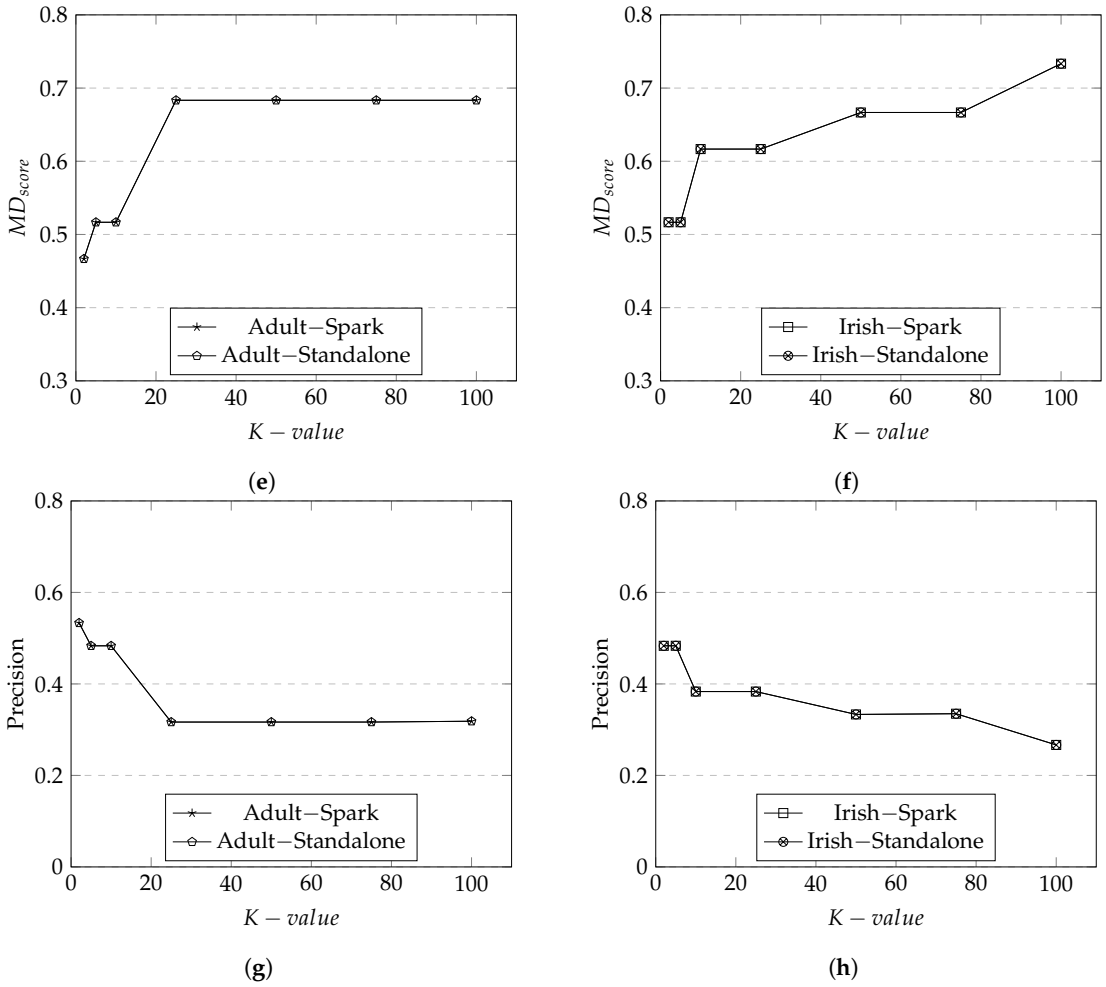

Figure 8. Data Utility Metrics for Adult and Irish datasets on both Spark and Standalone. (**a**) Discernibility Metric in Adult Dataset; (**b**) Discernibility Metric in Irish Dataset; (**c**) Average Equivalence Class in Adult Dataset; (**d**) Average Equivalence Class in Irish Dataset; (**e**) Minimal Distortion in Adult Dataset; (**f**) Minimal Distortion in Irish Dataset; (**g**) Precision Metric in Adult Dataset; (**h**) Precision Metric in Irish Dataset.

6.4. Scalability, Performance and Caching

We ran three sets of experiments to understand scalability, performance, and cache management as shown in Table 6. The execution time for running both FlatMapRDD and ReduceByKeyRDD was measured.

Table 6. Experimental Configurations for Scalability, Performance and Caching (*K-value* ∈ { 10, 20, 25, 50, 75, 100} on Spark).

#	Experiment	Anonymization Parameters	Dataset Size
1	*QID* Size	A set of | *QID* |∈ [1, 2, 3, 4, 5, 6] *	Adult = 10M
			Irish = 10M
2	Records Size	| *QID* | = 5 *	Adult = (5M,10M, 20M, 30M, 40M, 50M)
			Irish = (5M,10M, 20M, 30M, 40M, 50M)
3	Cache Storage Levels	| *QID* | = 5 *, Memory, Disk, Memory_AND_Disk, OFF_HEAP	Adult = 10M
			Irish = 10M

* It indicates the number of attributes that were used in the experiments.

6.4.1. Scalability

In the first set of experiments, we measure the scalability of SparkDA on Adult dataset and Irish dataset by varying the size of QIDs. Before running a scalability test, we first run an experiment for increasing the size of K group on a fixed number of QID 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. This can be explained by following. 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.

In contrast, as soon as we increase the size of QIDs, the execution time starts to increase. This is because the processing of QID involves applying generalization levels after counting for the number of distinct attribute values which require many iterative operations. Adding more QIDs involved generating more operations. Therefore, the execution time is increasing in the order of the increasing number of QIDs. This is shown Figure 9a,b.

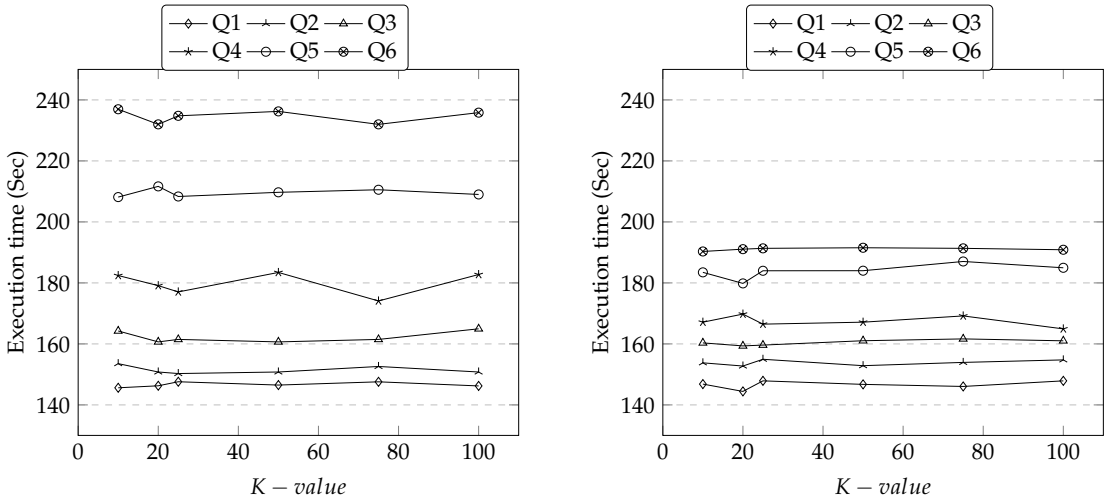

Figure 9. Execution Time vs. QID Size. (**a**) Adult Dataset; (**b**) Irish Dataset.

We examine the details of different QIDs from both datasets. It appears that there is a strong performance relationship between the distinctness of quasi-identifiers (i.e., often regarded as cardinality) and the execution time. For example, the execution time has sharply increased between Q4 and Q5 in Adult dataset. We observe that the new attribute "Occupation" in Q5 has a high cardinality and it affected the execution time. In addition, we see that higher execution times in Adult dataset as this dataset appears to have more variations of distinct values.

6.4.2. Performance

The second set of experiments is conducted to understand the performance of our proposal. We first compare the performance of our approach against existing data anonymization approaches. The list of existing approaches that were compared include: Spark based multi-dimensional sensitivity-based anonymization (Spark MDSBA) [18], MapReduce based multi-dimensional sensitivity-based anonymization (MR MDSBA) [15], Apache Spark based top-down specialization (Spark TDS) [17], and MapReduce based multi-dimensional top-down specialization (MR MDTDS) [15]. In order to ensure the comparability of results across different approaches, we used the same

workload and enforced our configuration to match with the experimental configuration discussed in References [15,17,18] as much as possible.

Figure 10 illustrates the execution time obtained across different methods. As clearly seen, our proposal outperforms other similar approaches by providing the lowest execution time. SparkTDS appears to show the highest execution time. Our analysis demonstrates that SparkTDS updates the score of all leaf which appears to be expensive additional overhead. This is because the increase in the number of leaves and associated operations (e.g., applying generalization level at leave) naturally demand more execution time especially for higher *K*-group sizes. The MapReduce-based approaches, seen in MR MDTDS and MDSBA, appear to have a higher execution time mainly due to expensive disk I/O associated with intermediate data. Spark MDSBA performs relatively well when compared to other approaches. We observed that Spark MDSBA uses a larger memory size compare to the dataset size which results in reduced execution time.

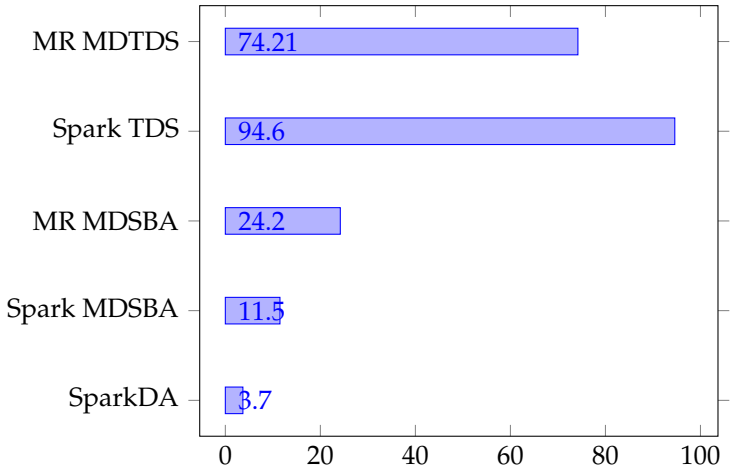

Figure 10. Performance comparison with existing approaches.

Secondly, we conducted a performance experiment to understand the impact of execution time against the growing number of records on the fixed size of 5 *QID* attributes. As seen in Figure 11a,b, the execution time remains same irrespective to the size of *K* group. 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 the 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 in both datasets.

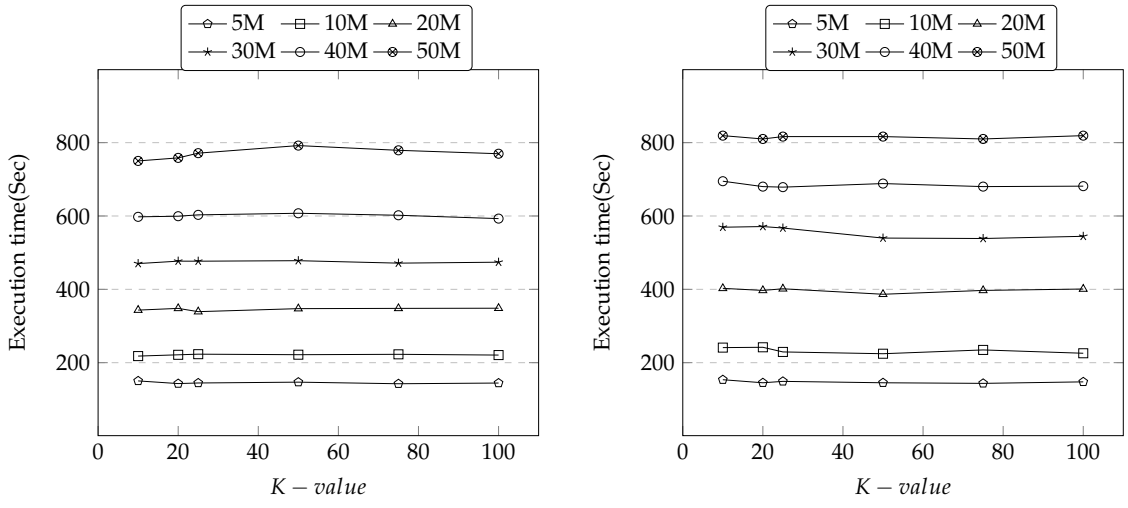

Figure 11. Execution Time vs. Record Size. (**a**) Adult Dataset; (**b**) Irish Dataset.

6.4.3. Caching

Spark offers a multiple cache storage levels to speed up the process of the same RDDs that are accessed multiple times. The Spark cache strategies can be categorized as follows [14].

- MEMORY_ONLY: RDD partitions are cached in memory only.
- OFF_HEAP: RDD partitions are cached outside the process heap (of JVM) in native memory therefore they are not processed by the garbage collector.
- MEMORY_AND_DISK: RDD partitions are cached in memory. If enough memory is not available, some RDD blocks (usually based on Least Recently Used, or other strategies [13] from memory are written off to disk.
- DISK_ONLY: RDD partitions are cached on disk only.

During the anonymization process, the two RDD transformations we utilize, FlatMapRDD and ReduceByKeyRDDs, are accessed multiple times for generalization from the main application SparkDA. We have set up our experiment with the different cache management options. The results are shown in Figure 12a,b. In general, the memory-based strategies where the RDD blocks are stored in the memory, such as MEMORY_ONLY and OFF_HEAP, outperformed compared to the cached in disk. Understandably, in-memory inside the JVM cache strategy MEMORY_ONLY took the least execution time compared to out of JVM memory cache strategy used by OFF_HEAP. The MEMORY_AND_DISK took more time than memory-based strategies but less than DISK_ONLY as expected as this strategy allows the switch from memory to disk when the allocated memory is fully consumed by RDD blocks. Comparing the overall cache performance, the average execution time for Irish dataset was less than Adult dataset. The higher generalization levels for different attributes in Adult dataset has contributed toward the increase in the execution time as there were more ReduceByKeyRDD operations for the generalization levels defined in the DGH thus the updates for attributes were more frequent.

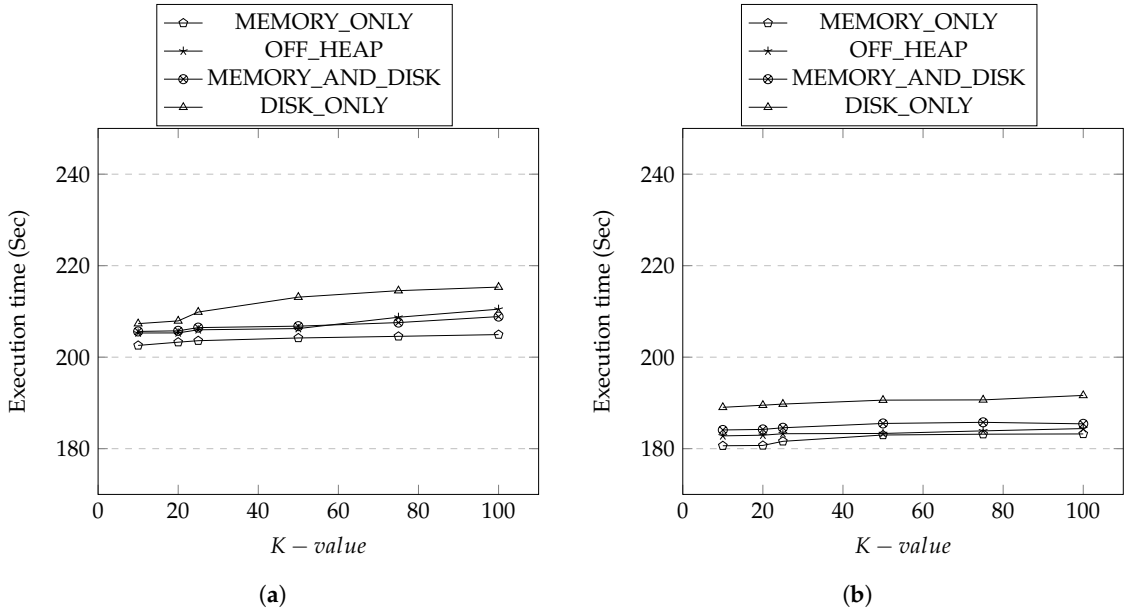

Figure 12. Execution Time vs. Cache Strategies. (**a**) Adult Dataset; (**b**) Irish Dataset.

7. Conclusions and Future Work

This work introduces "SparkDA" a new novel data anonymization approach designed to take the full advantage of Spark platform to generate privacy-preserving anonymized dataset in the most efficient way possible. Our approach is based on two RDD transformations FlatMapRDD and ReduceByKeyRDD with a better partition control, in-memory processing, and efficient cache management. These new innovations contribute towards reducing many performance overheads

associated in other similar approaches implemented in MapReduce. The set of experimental results showed that our proposal provides high performance and scalability while supporting high data privacy and utility required by any data anonymization techniques. We also provided insights of a set of performances associated with different memory management strategies offered by Spark and discovered that a side-effect could occur when there are too excessive demands to save data to executor's memory.

In future, we plan to extend our study to implement data anonymization strategy based on the subtree generalization scheme [1]. This new approach will solve the current limitation of the full-domain based generalization approach where attribution values are generalized equally without considering their respective parents' node which results in the loss of data utility to some degree. We also plan to extend our study to implement a more comprehensive data anonymization strategy for multi-dimensional datasets.

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; 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.; visualization, S.U.B.; supervision, J.J.-J.; project administration, S.U.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

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

References

1. Bayardo, R.J.; Agrawal, R. Data privacy through optimal k-anonymization. In Proceedings of the 21st International Conference on Data Engineering (ICDE'05), Tokoyo, Japan, 5–8 April 2005; IEEE: Piscataway, NJ, USA, 2005; pp. 217–228.
2. 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 2005, Tokoyo, Japan, 5–8 April 2005; IEEE: Piscataway, NJ, USA, 2005; pp. 205–216.
3. Sweeney, L. k-anonymity: A model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl. Based Syst.* **2002**, *10*, 557–570. [CrossRef]
4. Machanavajjhala, A.; Gehrke, J.; Kifer, D.; Venkitasubramaniam, M. l-diversity: Privacy beyond 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; pp. 24–24.
5. 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; IEEE: Piscataway, NJ, USA, 2007; pp. 106–115.
6. Kelly, D.J.; Raines, R.A.; Grimaila, M.R.; Baldwin, R.O.; Mullins, B.E. A survey of state-of-the-art in anonymity metrics. In Proceedings of the 1st ACM Workshop on Network Data Anonymization, Alexandria & Fairfax, VA, USA, 27–31 October 2008.
7. Sun, X.; Wang, H.; Li, J.; Truta, T.M. Enhanced p-sensitive k-anonymity models for privacy preserving data publishing. *Trans. Data Priv.* **2008**, *1*, 53–66.
8. Dean, J.; Ghemawat, S. MapReduce: simplified data processing on large clusters. *Commun. ACM* **2008**, *51*, 107–113. [CrossRef]
9. 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/Heidelberg, Germany, 2017; pp. 364–377.
10. 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/Heidelberg, Germany, 2017; pp. 88–99.
11. 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.

12. Grolinger, K.; Hayes, M.; Higashino, W.A.; L'Heureux, A.; Allison, D.S.; Capretz, M.A. Challenges for mapreduce in big data. In Proceedings of the 2014 IEEE World Congress on Services, Anchorage, AK, USA, 27 June–2 July 2014; IEEE: Piscataway, NJ, USA, 2014; pp. 182–189.
13. Zaharia, M.; Chowdhury, M.; Franklin, M.J.; Shenker, S.; Stoica, I. Spark: Cluster computing with working sets. *HotCloud* **2010**, *10*, 95.
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. Al-Zobbi, M.; Shahrestani, S.; Ruan, C. Improving MapReduce privacy by implementing multi-dimensional sensitivity-based anonymization. *J. Big Data* **2017**, *4*, 45. [CrossRef]
16. 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.
17. 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; IEEE: Piscataway, NJ, USA, 2017; pp. 4513–4521.
18. Al-Zobbi, M.; Shahrestani, S.; Ruan, C. Experimenting sensitivity-based anonymization framework in apache spark. *J. Big Data* **2018**, *5*, 38. [CrossRef]
19. 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, Heraklion, Greece, 3–5 May 2019; pp. 61–71.
20. 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/Heidelberg, Germany, 2019; pp. 646–662.
21. 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.
22. Nezarat, A.; Yavari, K. A Distributed Method Based on Mondrian Algorithm for Big Data Anonymization. In Proceedings of the International Congress on High-Performance Computing and Big Data Analysis, Tehran, Iran, 23–25 April 2019; Springer: Berlin/Heidelberg, Germany, 2019; pp. 84–97.
23. 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]
24. Sweeney, L. Achieving k-anonymity privacy protection using generalization and suppression. *Int. J. Uncertain. Fuzziness Knowl. Based Syst.* **2002**, *10*, 571–588. [CrossRef]
25. 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.
26. Ashwin, M.; Daniel, K.; Johannes, G.; Muthuramakrishnan, V. l-diversity: Privacy beyond k-anonymity. *ACM Trans. Knowl. Discov. Data* **2007**, *1*, 1–52.
27. LeFevre, K.; DeWitt, D.J.; Ramakrishnan, R. Mondrian multidimensional k-anonymity. In Proceedings of the 22nd International Conference on Data Engineering (ICDE'06) 2006, Atlanta, GA, USA, 3–7 April 2006; ICDE: Oslo, Norway, 2006; Volume 6, p. 25.
28. Li, J.; Wong, R.C.W.; Fu, A.W.C.; Pei, J. Anonymization by local recoding in data with attribute hierarchical taxonomies. *IEEE Trans. Knowl. Data Eng.* **2008**, *20*, 1181–1194.
29. Asuncion, A.; Newman, D. UCI Machine Learning Repository. Available online: https://archive.ics.uci.edu/ml/index.php (Accessed on 21 July 2020)
30. Central Statistics Office. This is Ireland: Highlights from Census 2011, Part 1. Available online:http://www.cso.ie/en/databases/ (Accessed on 25 July 2020)

31. 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.