Al-Zobbi *et al. J Big Data (2017) 4:45*
https://doi.org/10.1186/s40537-017-0104-5

Journal of Big Data

RESEARCH Open Access

Improving MapReduce privacy by implementing multi-dimensional sensitivity-based anonymization

Mohammed Al-Zobbi*, Seyed Shahrestani and Chun Ruan

*Correspondence:
m.alzobbi@westernsydney.edu.au
School of Computing, Engineering and Mathematics, Western Sydney University, Sydney, NSW, Australia

Abstract

Big data is predominantly associated with data retrieval, storage, and analytics. Data analytics is prone to privacy violations and data disclosures, which can be partly attributed to the multi-user characteristics of big data environments. Adversaries may link data to external resources, try to access confidential data, or deduce private information from the large number of data pieces that they can obtain. Data anonymization can address some of these concerns by providing tools to mask and can help with concealing the vulnerable data. Currently available anonymization methods, however, are not capable of accommodating the big data scalability, granularity, and performance in efficient manners. In this paper, we introduce a novel framework that implements SQL-like Hadoop ecosystems, incorporating Pig Latin with the additional splitting of data. The splitting reduces data masking and increases the information gained from the anonymized data. Our solution provides a fine-grained masking and concealment, which is based on access level privileges of the user. We also introduce a simple classification technique that can accurately measure the anonymization extent in any anonymized data. The results of testing this classification technique and the proposed sensitivity-based anonymization method using different samples will also be discussed. These results show the significant benefits of the proposed approach, particularly regarding reduced information loss associated with the anonymization processes.

Keywords: Anonymization, Big data, Data privacy, Granular access, Hadoop, MapReduce

Introduction

Conventional data uses anonymity techniques to hide personal information that can cause side link attacks. Anonymization methods, based on k-anonymity, have been widely employed for such purposes [1]. These methods fall into two broad categories. The first category constitutes of techniques that generalize data from the bottom of the taxonomy tree towards its top and are referred to as the bottom-up generalization (BUG) [2]. The second one is based on walking through the taxonomy tree from the top towards the bottom, known as the top-down specialization (TDS) [3]. However, traditional anonymization methods lack scalability and cannot, in general, cope with large sizes of data, as their performance degrades. Hence, a need for scaling up of the conventional methods is essential. Scaling up the conventional methods has resulted in several amendments to these fundamental techniques, including parallel BUG [4], hybrid BUG and TDS [5], and two-phase TDS [6].

All k-anonymity methods identify a set of attributes, known as quasi-identifiers (Q-ID). Q-IDs are attributes that contain private information, where adversaries can use to unveil hidden data by linking them to related external elements [7]. Current anonymization algorithms aim to improve data privacy by generalizing the Q-ID attributes through processes that utilize taxonomy tree, interval, or suppression. Q-ID attributes are replaced by taxonomy tree values or intervals. The best taxonomy or interval values are named as the best cut. In TDS method, the entropy is used to calculate the highest Q-ID score for the best cut. Other methods implement the median calculation to find out the best cut within the numerical Q-ID attributes [8]. However, these methods do not adequately address the scalability issues. Large data size is an obstacle for these techniques, as they need to fit the data into the limited memory that is available. Processing big data may overwhelm the hardware and result in performance inefficiencies even in parallel computing environments. To overcome such inefficiencies, some techniques split the large size of data randomly into small chunks of data blocks, so they can fit into the memory for carrying out the required computations [6]. However, this resolution does not provide a real scalability solution, since it degrades the performance and increases the masking of anonymized data. These points will be further discussed in "MDSBA groups definitions" section.

We introduce a novel anonymization method using BUG in k-anonymity that can address the scalability and efficiency issues. The method utilizes multi-dimensional sensitivity-based anonymization for data (MDSBA). The main aim of our method is to improve the anonymization process and to increase the usefulness of anonymized data. MDSBA saves computation time, by providing pre-calculated k value, pre-defined Q-ID for anonymization, and pre-calculated level of anonymization. To reduce heap memory usage, most current anonymization methods may split large data into small size blocks. The randomness associated with splitting data into blocks can result in reductions of the usefulness of the processed data. Our method merely splits data, and it is not based on random fragmentation of data. Our iterative split process provides a proper input for the parallel distributed domain. We fragment data horizontally in related to the class value, instead of random fragmentation, which increases anonymization loss. This process is an extension of the approaches reported in [8].

MDSBA method protects the direct record link and *Obvious Guess* attacks. In the direct record link attack, an adversary may use the Q-ID in a table, say, T, to re-identify the personal information, with the help of an external table, ET, which consists of similar Q-ID attributes. K-anonymity prevents such a linkage by ensuring at least some k records are entirely equivalent. In the *Obvious Guess* attack, an adversary may easily guess the sensitive attribute, without having to identify the record. Protecting from *Obvious Guess* attack will be further discussed later.

To elaborate on these points, the rest of this paper is structured as follows. The next part provides our motivations for proposing this anonymization approach and briefly explains its novelty. Next, we describe MDSBA structure and set the scene for the next parts. We then discuss the MDSBA vertical and horizontal grouping and equivalency in two sections. This is followed by a comparison between MDSBA and other anonymization methods. An example of MDSBA filtering and anonymization that is based on utilization of Pig Latin script is provided in the following section. The section after that presents the security solutions for the *Obvious Guess* attacks. In the last section, we present our experiments with the proposed MDSBA method along with the results and outcomes of our simulation studies are presented.

Enhanced anonymization: motivations and contributions

Many factors have resulted in more demands for big data analytics. There is a definite need for authorization techniques that can facilitate controlling the user's access to data in granular fine-grained fashions [9]. For instance, organization's alliance with business partners, contractors and sub-contractors may increase data access demands for analytics. Also, organizational structures impose variant needs for data access levels. The increasing demand for big data analytics and the lack of framework standardization have urged us to contribute to data analytics security in big data, by establishing a framework that can provide a granular access control for data analytics.

Our MDSBA's aim is the establishment of a framework that can provide a high scalability performance and a robust security protection against privacy violation in data analytics. For the sake of performance, the method is implemented by MapReduce operations. MapReduce is a framework for performing data-intensive computations and data analytics in parallel on commodity computers. Hadoop is one of the most prominent MapReduce frameworks that gained popularity in big data analytics. A MapReduce process operates to read and write by its file system, known as Hadoop distributed file system (HDFS). In Hadoop framework, two separate processes are performed; mapping and reducing, or MapReduce. The file system splits files into multiple blocks, and each block is assigned to a mapper that reads data, performs some computation, and emits a list of key/value pairs to a reducer process. Reducer, in turn, combines the values belonging to each distinct key according to some function and writes the results to an output file. Parallel and distributed computing is used for large data size, which may exceed 10's of Terabytes. The default file system's block size in Apache Hadoop version 3 is 128 MB [10]. Hadoop version 3's framework, known by YARN, is divided into a master server or NameNode, and slave servers or DataNodes [11, 12].

MDSBA algorithm is designed for Hadoop ecosystems such as Spark, Hive, and Pig. Traditionally, SQL language is the database dominator to manage and alter data. In Hadoop ecosystems, SQL-like programs replace traditional SQL database. For instance, Hive is the data warehouse system for Hadoop, which aims to simplify Hadoop usage for data workers by providing the SQL-like language for Hadoop [13]. Pig Latin is another Hadoop tool that manages warehouse system, by using a proprietary scripting language. Pig Latin treats data as a set of tuples, which supports tackling extensive datasets. Thereby, substantial parallelism and a slew of optimization techniques are supported. Pig provides customized support for a user defined function (UDF), by supporting many widely used languages, such as Java, Python [14], JavaScript, Ruby [15] or Groovy [16]. Similar to Hive, Pig supports ad-hoc queries, joins, and other SQL-like operations [17]. As shown in Fig. 1, Pig compiler resides at the top of Hadoop, which results in the client being responsible for running the script. Pig Latin is a combination notation of SQL-like and Java idiom. However, high-level programming notations cannot be implemented by Pig. Therefore, UDF is essential. MDSBA implements Pig scripts with UDF tool. However, Hive also can replace Pig for its similarity with Pig.

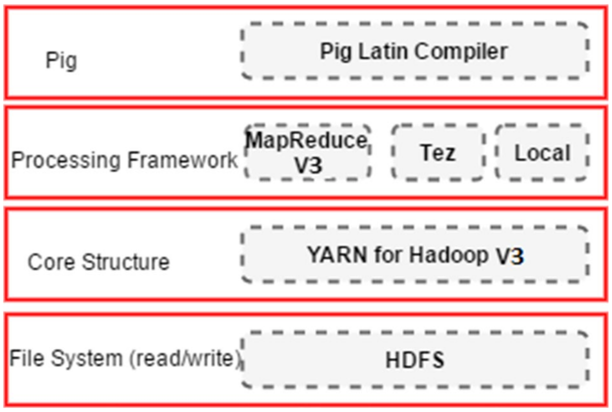

Fig. 1 Pig Latin structure in Hadoop v3: Pig compiler resides at the top of Hadoop, which results in the client being responsible for running the script. Pig is Hadoop ecosystem, and its compiler operates with YARN and HDFS. The framework contains three different processing modes; MapReduce, Tez, and Local

Introducing multi-dimensional sensitivity-based anonymization

MDSBA adopts Q-ID probability for applying masking of taxonomy tree, interval, or suppression. The probability value is derived from the taxonomy tree concept. The taxonomy tree T is propagated from the parent node w to some leaf nodes v. So, the probability of propagation from a parent node is $P(w) = \frac{1}{v}$. Figure 5 illustrates probabilities for each parent node in the education tree. The intervals, also, can be presented by probability values. If a number n was presented in an interval of minimum value V_{min}, and maximum value V_{max}, then the probability of obtaining that number within the interval range is $P(n) = \frac{1}{(V_{\max} - V_{\min})}$. For example, for the interval of [15–25[, the probability is (P = 1/10 = 0.1). This probability concept supports the fine-grained access for multiple users. For such cases, data is split into several groups or domains, and for each group, a different anonymization that depends on the user's access level, is applied. The anonymization process is managed by the value of sensitivity level ψ, which increases or decreases the information gained from the data. User with a higher value of ψ gain more information, and vice versa. The sensitivity level is determined by two major factors: the sensitivity factor ω and the aging factor τ. Consider a dataset, where k represents the k-anonymity value and $\overline{k}$ represents the ownership level, which is an instant of k-anonymity value. A large value of ownership level implies a weak ownership relation, so the weakest ownership relation is when $\overline{k} = k$. That is, the integer values of k and $\overline{k}$ are such that $2 \leq \overline{k} \leq k$. In other words, low values of $\overline{k}$ correspond to reduced anonymity as a result of higher ownership relations. The sensitivity factor ω is calculated based on its maximum and the minimum values. The maximum value of ω is defined as:

$$\omega_{\max} = \max\left(P\left(\text{qid}_0\right),\ P\left(\text{qid}_1\right),\ P\left(\text{qid}_2\right),\ P\left(\text{qid}_3\right)\right) \tag{1}$$

The minimum value of ω is defined as the product of all Q-IDs probabilities, or:

$$\omega_{\min} = \prod_{i=1}^{4} P\left[\text{qid}_i\right]. \tag{2}$$

Based on Eqs. 1 and 2, the value of ω can be found linearly between $\omega_{\min}$ and $\omega_{\max}$, as shown in Eq. 3:

$$\omega = \omega_{\min} + \left(k - \overline{k}\right)\left(\frac{\omega_{\max} - \omega_{\min}}{k}\right). \tag{3}$$

Equation 3 presents a linear relation between ω and $\overline{k}$ values. The equation lowers the value of ω when the user's $\overline{k}$ value is high, so that the data anonymity is higher. Users with less privileges are given higher $\overline{k}$, while the value of k is constant. We also propose that the lowest value of $\overline{k}$ should be two to avoid unique re-identification. The anonymization is only applied on external organization's access. External organizations are permitted to access an anonymized copy of the original data, while accessing the original data is exclusive to the data owners.

Equation 4 collates both terms of ω and τ to compute the sensitivity level ψ. The sensitivity level directly manages user's access privileges. Higher sensitivity levels lead to upper access ranks, where less data masking and concealments are applied. Also, the sensitive nature of an object can be made lower as its related dataset ages. The aging factor τ affects the sensitivity reversely, with considering the negative values of aging factor. Based on their owners' decisions, old objects can be deemed to be less sensitive, and increase ψ as they age. In essence, two factors determine the sensitivity level of the data, the sensitivity factor ω, and the aging factor τ, as described Eq. 4.

$$\psi = |\boldsymbol{\omega} + \boldsymbol{\tau}|. \tag{4}$$

Equation 4 is used to calculate the sensitivity level ψ, by establishing the sensitivity level of an object for a given user access level. The masking process tends to find any number close or smaller than the sensitivity value. For instance, if $\psi = 0.5$, then any value between 0 and 0.5 is acceptable. However, the values closer to ψ improve the information usefulness and the granularity precision. For instance, the age interval of [10–20[is derived from $\psi = 0.1$, while the age interval of [10–12[is derived from $\psi = 0.5$. We may notice the accuracy difference between both values of ψ, therefore, assigning the closer value to ψ is essential. The aging factor creates a perturbation to the sensitivity value. This factor becomes more significant as the data age becomes older than a particular age that we refer to as its obsolescence value. Equations 3 and 4 show that the lower sensitivity level requires a higher value of $\overline{k}$, which corresponds to lower information gain. In other words, the lower sensitivity levels, correspond to higher anonymization and masking levels of information.

The aging factor τ depends on four different parameters, the object obsolescence value $\varnothing$, the aging participation percentage ρ, the object age y, and the sensitivity factor ω. The $\varnothing$ value is defined as the critical age before the object sensitivity starts to degrade. It can be measured in units of hours, days, weeks, months or years, depending on objects obsolescence speed. However, $\varnothing$ cannot be given a value smaller than 2, so the value of 1 year, for example, can be replaced by 12 months instead. The aging participation percentage ρ is an approximation percentage chosen by data owners. It measures the aging factor participation in data objects. The aging factor τ can remain constant or decrease linearly. It remains unchanged when the age of the object y is less than its obsolescence value, so it remains constant if $y < \varnothing$. On the other hand, if the object age is greater than or equal to its obsolescence value, that is y ≥ Ø, then the sensitivity level increases logarithmically, which in turn decreases anonymization level. These two cases manage the aging factor τ, as described by Eq. 5.

$$\tau = \begin{cases} -\rho \times \omega & y < \emptyset \\ -\rho^{(y-2)} \times \omega \times \log y & y \geq \emptyset \end{cases} \qquad (5)$$

$$\emptyset \in \mathbb{Z}, \quad \emptyset \geq 0, \quad \text{and} \quad y \in \mathbb{Z} : y \geq 2$$

Data owners may set ρ to 0% if their data objects are not affected by time and age. Also, the maximum value of ρ cannot exceed the 90% to avoid a nil value for ψ. It can be noted from Eq. 5 that τ is always negative. In other words, the sensitivity level ψ increases with the age of the information and the passage of time. The newly created data objects result in a lower sensitivity level compared to the old data. In other words, if y < Ø then ψ < ω when the aging factor is incorporated. Equation 5 is derived based on our proposed ideas for incorporating the aging factor and sensitivity analysis for improving the anonymization process. It relies on a linear condition − ρ × ω and a semi-log component (− ρ(y − 2) × ω × logy). The linear part produces a constant value of aging factor when the information has not reached its obsolescence value. The semi-log portion is derived from the plotted graph, shown in Fig. 2. The figure illustrates the concept that incorporates the age of data objects. Older data is considered to have a logistic degradation with age and passage of time. The degradation starts swiftly before plateauing at the sensitivity factor value ω. To illustrate the time factor impact, let us consider the following example: a social media data has an aging participation value set at 0.9 with obsolescence value of 15 days and a sensitivity factor of 0.4 (ρ = 0.9, ω = 0.4, and Ø = 15). Plotting its sensitivity diagram can be initiated by assuming a constant value of sensitivity factor multiplied by the aging participation percentage to find out the aging factor. Therefore, τ = − ρ × ω = − 0.36, and ψ = 0.04. Next, a logistic graph can describe the sensitivity level direct proportions the variable y increase. The sensitivity

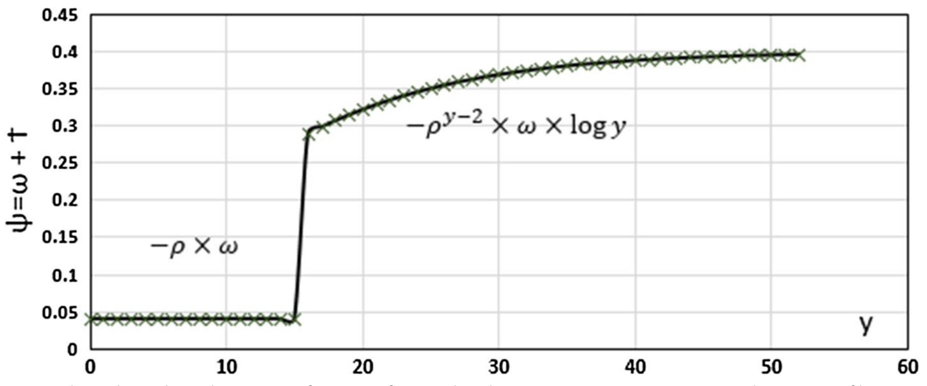

Fig. 2 Plotted graph to derive Eq. 5 for aging factor: This diagram presents our suggested concept of how the sensitivity value would look like with the aging factor. From this diagram, a mathematical equation was derived as $\tau = \begin{cases} -\rho \times \omega & y < \emptyset \\ -\rho^{(y-2)} \times \omega \times \log y & y \geq \emptyset \end{cases}$ If data owners found that his data importance degrades with the time, then the degradation should start after a period of time, named as obsolescence value Ø

level increases dramatically between days 15 and 30. This sharp increase describes the object degradation importance, which will intentionally reduce the obscurity level on data anonymization.

The following example illustrates the calculation steps of sensitivity value ψ. Consider an object with three Q-ID attributes, say student IQ test results, similar to those shown in Table 1. The data owner intends to anonymize the data with $k = 20$, the obsolescence value $\emptyset$ is set to 10, the aging participation ρ is 70%, and the age of the object y is13 years. Now, suppose a user was given an ownership value $\overline{k} = 10$. Based on the given attributes, the sensitivity level ψ can be calculate using Eqs. 4 and 5. To find out the sensitivity factor ω, it can be noted that the values of $\omega_{\max} = \max\ (0.01, 0.005, 0.125)$ is 0.125 and $\omega_{\min} = 0.01 \times 0.005 \times 0.125$ is 6.25×10^{-6}. The value of ω as per Eq. 3 is $\omega \approx 0.063$. Based on Eq. 5, the aging factor $\tau = -\ (0.7)\ 11 \times 0.063 \times \log\ (13)$ can be seen to be $-\ 0.00138$. Therefore, the sensitivity level $\psi = 0.063–0.00138$ is found to be 0.062.

The previous mathematical equations support the granularity of access control. Our proposed MDSBA is a complete framework that can control both ends of federation service and service providers. Security assertion markup language (SAML) v2 is used to map user's access level between both ends. The framework consists of three primary services; Core, Initializer, and Anonymizer. The Core service embeds XML files as assertions to transmit the user's authentication results parallel with the user's ID, and access level. The access level is determined by delegated roles, while roles are mapped to Q-ID groups. The Initializer service provides initial documents to prepare data for anonymization. The initializer also generates the Pig script line by line, by referring to the provided XML files. Eventually, the Anonymizer service executes the anonymization script remotely. The Initializer operates in the service provider gateway, which is not a member of Hadoop domain. Complete details about the MDSBA framework are further explained in this paper [18].

MDSBA groups definitions

It is evident that records equivalency, in most datasets, increase in parallel with data growth [19]. Therefore, splitting data nominally may increase the data equivalency. Random data split ignores the distribution of equivalent records amongst tables. For instance, in Adult data, patient's age, sex, and education attributes may appear similar in the first one thousand records, and then related records may appear again at the end of the table. Hence, the random appearance of similar records may reduce the number of equivalency on splitting data into small random chunks, which increases data masking.

Table 1 Three Q-ID attributes example

Q-id (attribute)	Q-id type	Probability
Q-id0 (IQ_value)	Interval IQ_value = [50–150]	P(Q-ID0) = 1/(150 − 50) = 0.01
Q-id1 (Student_country)	Taxonomy tree Student_Country_Level1 = {German, French, Chinese, Kenyan, American…} Student_Ancestry-Level2 = {Caucasian, Asian, Middle Eastern, African, Red Indian…} Q-id_level-3 = {human}	P(Q-ID1-L2) = 1/150 = 0.007 P(Q-ID1-L3) = 1/200 = 0.005
Q-id2 (Student_Grade)	Suppression Student_Grade = {A+, A, A−, B+, B, B−, C+, C, C−, D+, D, D−, F}.	P(Q-ID2) = 1/13 = 0.077

The following three definitions describe the horizontal split of data by implementing the grouping method of equivalency in MDSBA. We believe that splitting data nominally will increase the equivalency ratio. Suppose a dataset D, with a total number of m attributes, and n records. The data owner defines the number of quasi-identifiers as Q in m attributes, so $Q \leq m$. Let some q attributes have equivalent records k, where q attributes are part of Q, so $q \leq Q \leq m$. Hence, the wholly or partly similar records are defined as follows:

Definition 1 All D records are split based on the class C values. Each set of records that contains similar class value is aggregated in a G_i group, where i denotes the number of values appear in the class C. Every G_i group is further processed individually.

Definition 2 The fully-equivalent group (SG) contains some k equivalent records, in some Q attributes. For this group, there is no any anonymization process applied.

Definition 3 The Semi-equivalent group (SSG) contains some k equivalent records, in some q attributes, where $2 \leq q \leq Q - 1$. The highest Q-IDs probability is usually chosen for q attributes equivalency. The anonymization is applied on the rest of non-equivalent Q-IDs.

Definition 4 The non-equivalent group (NG) contains a number k of equivalent records, in some q attributes, where the number of $q = 1$. The highest Q-ID probability is usually chosen for q attribute. The anonymization is applied to the rest of non-equivalent Q-IDs.

As explained in Definitions 3 and 4, the SSG and NG anonymization is applied on the lowest Q-ID probabilities. For instance, in Adult data the Q-IDs probabilities are; $P[Age] = 0.01$, $P[Sex] = 0.5$, $P[Edu] = 0.08$. Based on these values, the SSG anonymization will be applied on the Q-ID with the lowest probability, which is [Age]. So the semi-equivalency is measured by grouping Sex, and Edu attributes, while Age attribute is anonymized as per interval. The NG anonymization will be applied to all Q-IDs except the one with the highest probability. In our example, the anonymization will be applied to [Edu] and [Age], while records are grouped as per [Sex] equivalency.

MDSBA framework splits data recursively to create three dependent stages of MapReduce processes. Stage one aims to produce Gi groups for the whole dataset with one Pig command only, which filters each data record based on its class attribute, as described in Definition 1. The number of the produced Gi groups relies on the number of sensitive class values. For instance, Table 3 contains four values of sensitive attribute (Cancer); these are {no positive histology, Positive histology, Positive microscopic confirm, and Positive laboratory test}. The four values create four groups of Gi = {G0,G1,G2,G3}. Each group is stored in a separate HDFS folder. Stage two reprocesses G groups in parallel, to categorize data records between wholly or partly equivalent or between SG, and SSG. According to definitions 2 and 3, the SG groups denote the equivalent Q-ID group, while the SSG groups denote Semi-equivalent group. The SG groups are equivalent, so anonymization is skipped, and data is stored in an output directory. The third stage's input is derived from the second stage's output, so SSG groups are stored in an input directory waiting for stage three. The process starts by reading SSG groups and abstracting the non-equivalent groups, denoted by NG. Both of NG and SSG are anonymized by using user defined function Java program.

In Adult data example, Q-ID attributes are grouped by the three Q-IDs (Age, Sex, and Edu), so the number of equivalent records must be greater than or equal to $\overline{k}$. These equivalent records are stored in SG, while the equivalent records with a number smaller than $\overline{k}$ are stored in SSG. In the second stage, the three attributes are grouped again by the largest probability values (Sex, and Edu). The equivalent records with a number greater than or equal to $\overline{k}$ will be anonymized by Java program. The anonymization is applied on the attribute with the lowest probability value, which is Age. The equivalent records with a number smaller than $\overline{k}$ are stored in NG, in order to be further grouped and anonymized.

MDSBA and quasi-identifiers groups

MDSBA presumes a table model with a set of Q-IDs and one sensitive attribute (S), known by a Class. For instance, a table of attributes = {AGE, RACE, STATE, ADDRESS, DISEASE} can be divided into Q-ID = {AGE, RACE, STATE}, and one class of the sensitive information S = {DISEASE}.

Definition 5 A table T contains a Q number of Q-IDs, denoted by Q(Q-ID), and a C number of classes, denoted by C(S). Attributes are grouped vertically by dividing Q and C attributes into groups. Each group contains one class attribute and some two to four Q-ID attributes. In other words, $2 \leq$ Q(Q-ID) ≤ 4, and C(S) $= 1$.

MDSBA aims to split data nominally to reduce the data size for the anonymization process by vertically bundling some Q-IDs and Classes into small groups. Many datasets may contain more than one class attribute and a considerable number of Q-IDs. The increased number of Q-IDs may produce a massive computation cost and data overflow, which may unexpectedly terminate the program. Also, bundling small groups of Q-IDs assists the management of access control and authorization. Hence, this technique supports the anonymization performance and scalability. The Q-ID groups are determined by data owners and can be divided randomly or logically. For instance, Seer Cancer Data contains the following attributes = {Sex, Year of Diagnosis, County, State, Grade, Diagnostic Confirmation, Race, Age}. If the data owner decides to divides attributes to Q-IDs and Classes, then he/she may assign six Q-ID attributes and two categories. Thereby, having two bundles of Q-IDs and one class for each bundle can be a combination of three by three or four by two of Q-IDs. As an example of bundle 1 = {Sex, Race, Age}, and bundle 2 = {year of diagnosis, County, State}, while the classes are; {Grade, Diagnostic Confirmation} one per each bundle.

Comparing MDSBA with other anonymization methods

Similar to SQL, SQL-like languages implement multi-technique to handle database queries, such as temporary tables, indexing, and temporary files. SQL-like applies efficient techniques to increase the speed and performance on processing a large data size [20]. In contrast, programming languages, such as Java or C, implement arrays or array lists to contain data temporarily during the processing tasks and iterations. Programming languages use a limited temporary memory (RAM) during operations, while SQL-like operations are conducted between temporary memory and large repository storage (disk). The SQL-like structure is essential in big data. Implementing programming languages causes high processing costs and low scalable operations. SQL-like also facilitates the handling of large data sizes and leverages the scalability during the analytics process. Anonymization is an example of high-cost operations that needs SQL-like to relieve data overflow over the memory limitations [21].

MapReduce ecosystems, such as Pig and Hive, implement SQL-like as a prominent solution for data flow platform. Apache Pig provides nested data types like tuples, bags, and maps that are not available in MapReduce. Pig provides many built-in operators to support data operations like joins, filters, orders, sort, and others. Whereas performing the same function in MapReduce is a complicated task. Pig compiler manages the data flow by creating a group of tuples or bags. The data group is emitted to YARN container for further processing. If the data group exceeded the memory size, then it spills the retained data to the disk. This kind of management cannot be applied efficiently on the current anonymization methods, such as MDTDS. The reason for that is the limitations of Hadoop ecosystems on dealing with large data flow with multiple conditional iterations. Pig, for instance, implements UDF to overcome the Pig programming limitations. However, the UDF functions operate in external JVMs, which are not related to YARN platform. UDF transmits data to a black-box that is not able to apply parallelism and a proper data management. Usually, the local JVM may fail to utilize the large size of data in memory, as a reason of the limited Java Heap memory. Therefore, developers should tackle this concern by reducing the UDF use, and minimize the data size processed by UDF [22].

Figure 3 compares between MDSBA and TDS algorithms. The comparison shows the absolute dependence of TDS algorithm on UDF. In TDS, the conditional iteration is necessary to find out the best Q-ID score to be specialized. Also, the level of specialization is unknown, so a continuous iteration is needed before finding the best cut. On the contrary, MDSBA pre-determines the Q-IDs that needed to be anonymized, and the level of anonymization. This saves iteration and computation times. Moreover, MDSBA reduces the data size processed by UDF. This is essential to avoid Java Heap memory error, which may unexpectedly terminate the anonymization process.

Splitting data by applying the (filter) command is necessary to distribute the massive data load among nodes. The anonymization algorithm should consider the data flow sequence, by choosing the SQL related commands, such as filter, group, and join. MapReduce cluster nodes serialize large size of the data blocks before anonymizing data. The anonymization algorithm could be a bottleneck for such a large block of data if data were emitted to UDF. Moreover, the continuous iteration process with a large data size will degrade the anonymization performance. However, the positive side of the iteration is producing a high accuracy result, which in turn increases the information gained. In MDSBA, the pre-defined parameters of Q-ID attributes, and the anonymization level may result in less information gain. However, in big data, applying such complicated processes may not affect the final results of statistical output. The small statistical values can be even ignored since the statistical results follow the principle of estimation

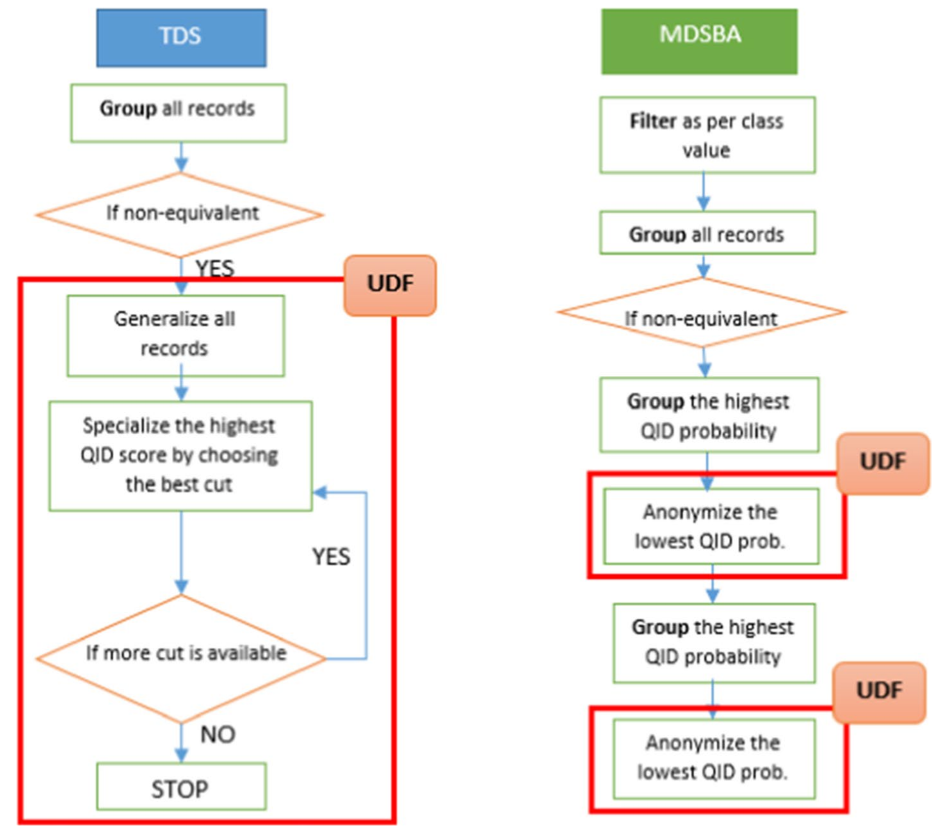
TDS
Group all records
If non-equivalent
YES
UDF
Generalize all records
Specialize the highest QID score by choosing the best cut
YES
If more cut is available
NO
STOP
MDSBA
Filter as per class value
Group all records
If non-equivalent
Group the highest QID probability
UDF
Anonymize the lowest QID prob.
Group the highest QID probability
UDF
Anonymize the lowest QID prob.

Fig. 3 Comparison between the anonymization algorithms of MDSBA and TDS. This figure compares between both anonymization methods, by showing each algorithm's dependency on UDF. The figure shows a large dependence by TDS, and a low dependence by MDSBA

prospect. This gives data miners a flexibility of approximating and rounding some numbers to a few decimal places [23]. Therefore, pre-calculating the k value, and pre-determining the attributes needed to be anonymized is an advantage. This non-accuracy will not dramatically affect the data analytics results.

MDSBA implementation example

Let us study the following complete example of anonymizing data by implementing Pig Latin script. We will apply anonymization on Seer Cancer Data. Table 2 shows the suggested Q-IDs, and the probability for each attribute and Table 3 shows the Class attributes with four sensitive values.

Suppose that data owner assigns $k = 400$, and aging factor $\tau = -0.02$. A user attempts to access this data with an ownership level $\overline{k} = 250$. Referring to Table 2, and Eqs. 1–3, the $\omega_{max} = 0.33$, $\omega_{min} = 2.5 \times 10^{-06}$, so the sensitivity factor is found to be $\omega = 0.124$.

Table 2 Seer Cancer Data Q-IDs and probability

	Year of diagnosis [int]	County [int]	Race [chararray]	Age [int]
Data	1973–2012	80 county	Black, white, others	0–85
Probability	0.025	0.013	0.33	0.012

Table 3 Seer Cancer Data class (four sensitive values)

Diagnostic
No positive histology
Positive histology
Positive microscopic confirm
Positive laboratory test

The data will be anonymized with a sensitivity level $\psi = \omega + \tau = 0.124 - 0.02 \approx 0.122$. This sensitivity level indicates anonymization degree that will be applied on attributes. For instance, if four numerical Q-ID attributes need to be anonymized, then each attribute can be rewritten by a minimum interval distance of 2. Hence, the factorial result for the four attributes is $\frac{1}{2} \times \frac{1}{2} \times \frac{1}{2} \times \frac{1}{2} = 0.063$, which is accepted value since it is smaller than $\psi = 0.122$. Therefore, if the anonymization was applied on the four numerical Q-IDs, then the year of diagnosis can be given intervals of 2 such as (1973–1975), and similar interval can be applied to the rest of the anonymized attributes. However, the interval distance is determined based on the minimum and maximum numerical value, within the semi-equivalent group. This interval is determined by the UDF program.

Figure 4 is divided into five main stages. Each stage generates a set of groups. Stage Zero aims to flesh out records related to the *Obvious Guess*. Stage One filters or splits

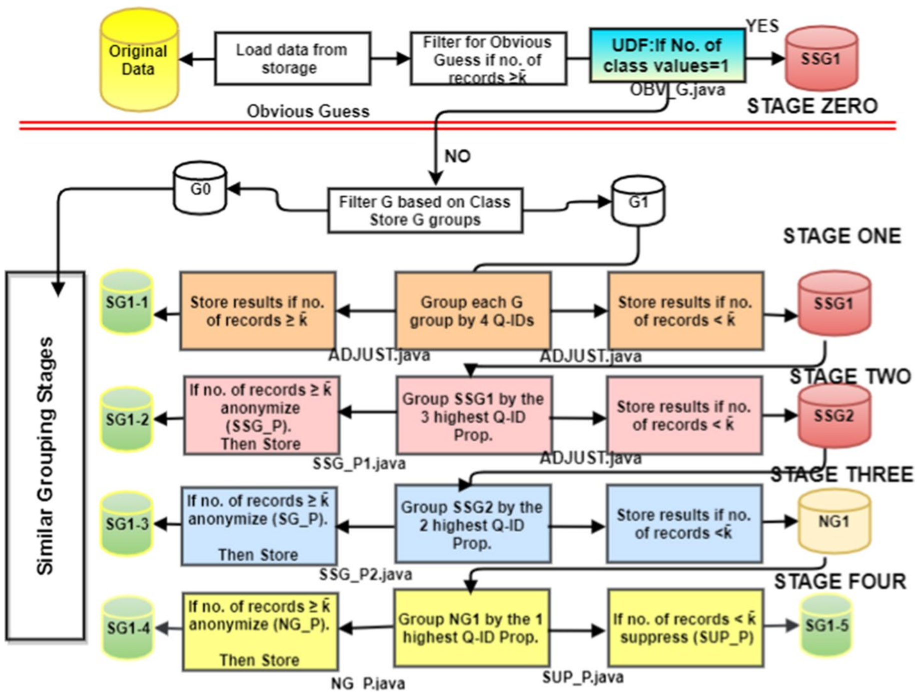
Original Data
Load data from storage
Filter for Obvious Guess if no. of records ≥k̄
UDF:If No. of class values=1
OBVI_G.java
YES
SSG1
STAGE ZERO
Obvious Guess
NO
G0
Filter G based on Class Store G groups
G1
STAGE ONE
Similar Grouping Stages
SG1-1
Store results if no. of records ≥ k̄
Group each G group by 4 Q-IDs
Store results if no. of records < k̄
SSG1
ADJUST.java
ADJUST.java
STAGE TWO
SG1-2
If no. of records ≥ k̄ anonymize (SSG_P). Then Store
Group SSG1 by the 3 highest Q-ID Prop.
Store results if no. of records < k̄
SSG2
SSG_P1.java
ADJUST.java
STAGE THREE
SG1-3
If no. of records ≥ k̄ anonymize (SG_P). Then Store
Group SSG2 by the 2 highest Q-ID Prop.
Store results if no. of records <k̄
NG1
SSG_P2.java
STAGE FOUR
SG1-4
If no. of records ≥ k̄ anonymize (NG_P). Then Store
Group NG1 by the 1 highest Q-ID Prop.
If no. of records < k̄ suppress (SUP_P)
SG1-5
NG_P.java
SUP_P.java

Fig. 4 Anonymizing Seer Data by using Pig Latin with four Q-ID attributes. This figure represents our MDSBA method structure. It depends on iterating processes of grouping data. If a group of four Q-ID attributes are chosen. Stages of grouping by four Q-IDs, then by three Q-IDs, then by two Q-IDs and finally by one Q-ID. The grouping chooses the highest three Q-IDs probability, then highest two Q-IDs probability, and finally the highest one probability. In each stage, the groups are successful when the number of equivalent or semi-equivalent records is greater than $\bar{k}$. The records < $\bar{k}$ are ungrouped, and then regrouped again

data based on class values to generate G groups. This is implemented by loading data from HDFS location, before the splitting, so each class value is assigned to one G group. Each G group is then stored separately. After creating G groups, a (group by) command is applied to all Q-ID attributes for each G group. In the four Q-ID attributes case, four Q-ID attributes are grouped for equivalency. The idea is grouping all Q-ID attributes to filter the entirely equivalent records and store them in SG location. The non-equivalent records are stored in a separate SSG1 location. In stage two, three Q-ID attributes of SSG1 data will be grouped for semi-equivalency. The chosen three Q-IDs for grouping have the highest Q-ID probabilities. The records that pass the semi-equivalency criteria are further anonymized. The anonymization is applied on the lowest probability value of Q-ID attribute and finally, stored in SG location. The records that fail the semi-equivalency criteria are stored in an SSG2 location for the next stage. In stage three, a similar concept is applied to a semi-equivalent group, before applying anonymization. The grouping command will be implemented for the highest probability values of two Q-ID attributes, while the lowest two probability values are anonymized. However, in all previous stages, the grouping command is cumbersome. The command alters the records format and transposes data from horizontal to vertical. Hence, the grouped records cannot be re-grouped [21]. Therefore, we need a program that can adjust the grouped records back to their original format. A UDF Java program reads the data as bags and converts them back to tuples, named (ADJUST.java). All grouping processes are filtered by comparing their number of records with the $\overline{k}$ value, if it is larger, then an anonymization will take a place. Stage two filters three Q-ID attributes of SSG1 data, so if cnt $\geq \overline{k}$, then SSG_P1 Java program will anonymize data before storing it in SG group. Otherwise, data is stored in SSG2 group for the next stage. In stage three, SSG2 data will be grouped by the largest two Q-ID attributes. Data, then, will be anonymized by SSG_P2. In the final fifth stage, records are grouped by the highest QID probability and anonymized either by NG_P if $\geq \overline{k}$, or by SUP_P if $< \overline{k}$. A complete Pig script for Adult data is available in Appendix 1, with three Q-ID attributes.

Six UDF Java programs are defined in this process: SSG_P1, SSG_P2, NG_P, SUP_P, OBV_G and ADJUST. The first three programs anonymize data by one Q-ID attribute as in SSG_P1, or by two Q-ID attributes as in SSG_P2, or by three Q-ID attributes as in NG_P. The program SUP_P suppresses all Q-ID attributes as a last resort, where one Q-ID grouping does not meet the k-anonymity criteria. The program SSG_P2 is only used when the number of Q-ID = 3. The program NG_P may anonymize 1, 2, or 3 attributes, when the total number of Q-IDs is 2, 3, or 4 simultaneously.

Protection from Obvious Guess

In the *Obvious Guess*, an adversary may easily guess the sensitive attribute (class), and without having to identify the record. For example, in Seer data, we may have a data bag shown in Table 4. In the described bag example, the diagnostic may contain one value only. This class demonstrates the *Obvious Guess* breach, which can be interpreted as; any black person lives in the county 125, and was diagnosed on 1997, with an age of 25 must have a positive histology cancer. The *Obvious Guess* occurs if the bag contains only one class value.

Table 4 ***Obvious Guess*** **example**

Year of diagnosis	County	Race	Age	Diagnostic
1997	125	Black	25	Positive histology
1997	125	Black	25	Positive histology
1997	125	Black	25	Positive histology

MDSBA applies the security protections before executing any grouping operations. The process can be divided into three sub-processes. Initial verification is executed in Pig Latin. The steps include, grouping all Q-IDs, while the class attribute is excluded, and transferring any group number $\geq \overline{k}$ to UDF program. The UDF executes the rest of the two sub-processes. These sub-processes are naïve and should not cost extra RAM. The first statement is if statement, to find out if a single class value is found. If yes, then store data in a temp_file over HDFS. If more than one class value is found, then store data in SSG group, so it will ready for Stage two. As described in Fig. 5. Our goal is reducing the number of arrays and iteration processes inside UDF Java program. This is important concept, since Pig Latin was designed to spill out of JVM memory to the disk. However, Java program cannot spill to the disk, so Java heap memory may terminate the program. Hence, we kept the processing cost, with this massive data flow, to the minimal level.

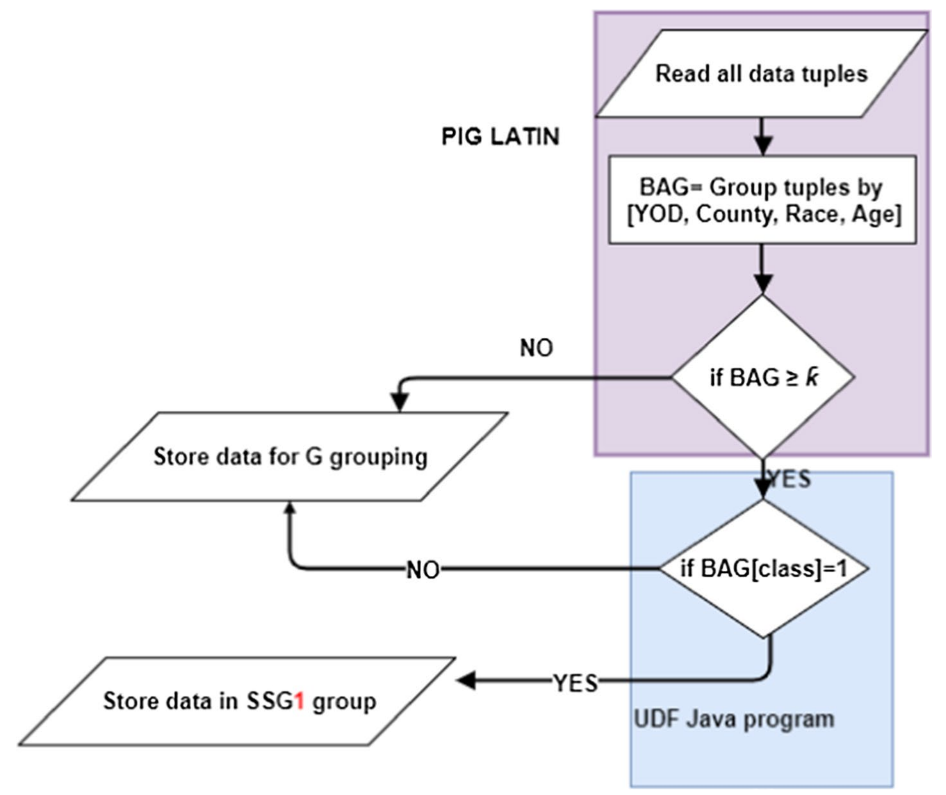

Fig. 5 *Obvious Guess* filtration for Seer data. One of the challenges in MDSBA is the "*Obvious Guess*". This happens in fully-equivalent records only. Stage zero filters the *Obvious Guess* and moves it directly to Stage two. If found that full-equivalent records have only one class value, then this group is defined as "*Obvious Guess*", and must be stored in SSG group ready for stage two

Anonymization classification

Many techniques have been implemented to measure the performance of the anonymization methods. Techniques, usually compare data before and after anonymization to ascertain the information loss. One of the most widely used approaches relevant to information loss is based on measuring the entropy of data after anonymization by calculating the tuples fraction in each block of anonymized data, as in Eq. 6.

$$E(T[v]) = \sum_{s \in S} T[qi, s] \log\left(T[qi, s]\right) \tag{6}$$

where $T(qi, s)$ is the tuples fractions in qi blocks and s denotes the sensitive attribute.

The entropy then is used with the score equation $\text{Score}(v) = \frac{\text{InfoGain}(v)}{\text{AnonyLoss}(v)+1}$, which defines the relation between information gained and anonymization loss. As shown in Eq. 7, the information gained adopts the entropy equation [24].

$$\text{InfoGain}(v) = E(T[v]) - \sum_{c} \frac{|T\ [c]|}{|T\ [v]|} E(T[c]) \tag{7}$$

The score equation was developed for multi-dimensional top-down specialization (MDTDS), where data blocks are specialized or generalized by implementing Eqs. 6 and 7. Other techniques implement the decision tree by building a cost matrix to find out the classification costs and errors. This technique is usually executed by Naïve Bayes classifier or by C 4.5 classifiers [25]. However, these techniques predict decisions before and after anonymization instead of finding the actual degradation of data usefulness after anonymization.

We adopted an alternative naïve equation that measures the percentages of information loss after anonymization, denoted by Disruption ($\mathcal{D}$). The disruption value is a benchmark that gives a general indication of the size of anonymization loss. As shown in Eqs. 8 and 9, each anonymized block of tuples is calculated individually, and finally, the $\mathcal{D}$ value is the result of the total summation of all anonymized blocks s. Each block of data is a data bag produced by grouping a set of tuples. Let us assume that an anonymized block of data contains some M records, in a total number of N dataset records.

$$\mathcal{D}\text{s} = \frac{M}{N} \times \frac{0.01}{\prod P[\text{QID}]} \tag{8}$$

where $\prod$ P[QID] is the factor of Q-IDs probabilities used in anonymizing each block.

$$\mathcal{D}[\text{total}] = \sum_{s=0}^{i} D_s \tag{9}$$

Equation 8 is derived from the reverse proportion between the Q-ID probability and $\mathcal{D}$. The $\mathcal{D}$ value increases with the increasing number of Q-ID attributes that participate in anonymization. Hence, implementing three anonymized Q-IDs will result in a higher $\mathcal{D}$ value than implementing two anonymized Q-IDs. The following example illustrates

Eqs. 8 and 9. Recalling the adult data, and considering the total number of records is 500. Two blocks of data were anonymized by two Q-IDs of education and sex. The number of anonymized records for these two blocks was 15 and 30 respectively. The education anonymization was given level 2, which relates to (certificate) and (degree), respectively. Based on the EDU taxonomy tree, as shown in Fig. 6; the first block probability is $P[\text{EDU}] = 0.067$ and the second block probability is $P[\text{EDU}] = 0.17$. Also, the SEX probability is $P[\text{SEX}] = 0.5$. Based on the previous two equations and the given information, the value of $\mathfrak{D}$ is calculated as $\mathfrak{D}1 = (15 \times 0.01)/(500 \times 0.067 \times 0.5) = 0.009$, and $\mathfrak{D}2 = (30 \times 0.01)/(500 \times 0.17 \times 0.5) = 0.007$. Referring to Eq. 9, the total value of $\mathfrak{D} = 0.016$.

Validation and simulation analysis

Our first objective is to evaluate the level of information loss in MDSBA and to compare it with the other anonymization methods. All experiments were performed on our university Hadoop lab, which includes four virtual machines, with one NameNode and three DataNodes. Each node's CPU is Intel(R) Xeon(R) CPU E5-2665 0 @ 2.40 GHz × 86_64, with a physical memory of 8 GB. The operating system is CentOS 7 configured with Hadoop version 2, and Pig Latin version 0.15.0. Four different common data were used in our experiments including, adult data, Seer Cancer Data, Heart Disease Data, and Kasandr dataset. Each dataset was randomly enlarged up to three size varieties; these are 1.2, 3.3, and 4.6 GB. The enlargement was created by applying Excel VBA script to produce tens of (.csv) files.

Seer Cancer Data of Q-IDs and the class are described in Tables 2 and 3. Adult data Q-IDs = {Age, Edu, Sex} and the class is salary = {<= 50 K, > 50 K}. Also, Heart Disease Data Q-ID = {Age, Sex, Smoker, cp}, and the class of electrocardiographic results, i.e., restecg = {normal, having ST-T wave abnormality, and showing probable

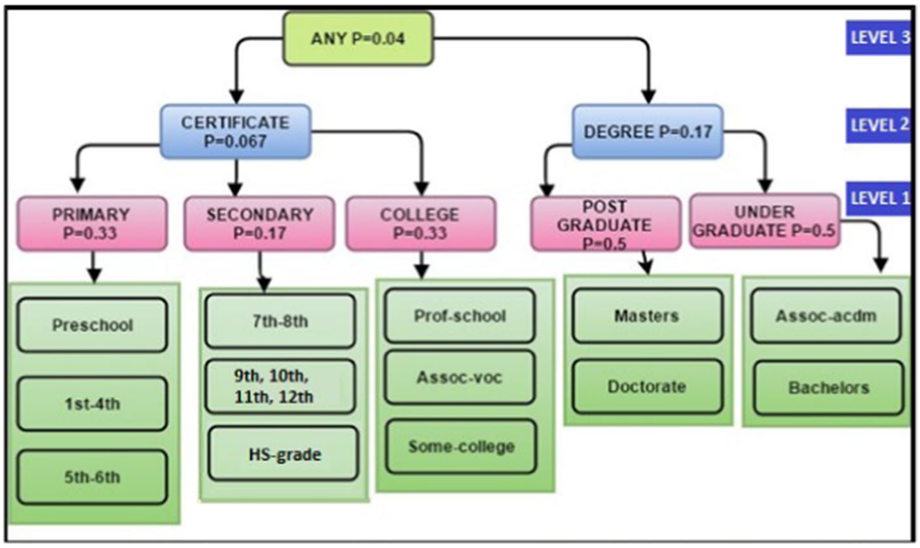

Fig. 6 Taxonomy tree for EDU in Adult data. The taxonomy tree and probability calculation for each node, child, parent, and root. This tree was designed for education level that was addressed in census data

or definite left ventricular hypertrophy}. The Kasandr dataset consists of the following attributes = {userid, offerid, city, category, merchant, purchase_date, implicit_feedback}. For security reasons, the user id is omitted, and the chosen Q-ID = {city, category, purchasedate}. The class is divided based on the 738 types of the merchant. The dataset of Kasandr was collected in Germany. The cities of Germany are around 80 cities, and the products category is around 50 types, while the purchase date was recorded for 5 years over 365 days per annum. Based on these numbers the probabilities are given as; city = 0.0125, category = 0.02, and the date = 0.0005.

In our experiments, we implemented Java as a UDF combined with Pig scripts. The script reads the database by using Hadoop reading process through HDFS. We divided our three experiments into three sections; MDTDS, BUG, and MDSBA. Both BUG and MDTDS methods were conducted by coding a Java program embedded within Pig script as a registered JAR. However, both methods, BUG, and MDTDS, were executed several times to avoid Hadoop job failure. To prevent this failure, we split the large file gradually into smaller files to overcome the unexpected error occurrences. There was no need to split the large files when the value of $k < 16$. The number of splits has increased parallel with the increasing value of k. The data split for 4.6 GB dataset is illustrated in Table 4. A separate task processes each small data file. The split is essential to reduce the data overflow across the UDF program. BUG method performed better with a more significant data size, as shown in Table 5.

Our first experiment implements a commonly available classifier for comparison between the three anonymization methods. The first comparison relies on Naïve Bayesian classifier. We applied anonymization on two datasets, Adult and Seer datasets. After completing the anonymization processes for each dataset, we first calculated the classification error by using RapidMiner Studio. The calculation was conducted before and after the anonymization. We implemented similar testing concept as in [3]. The classification error before anonymity is called Baseline Error and denoted by BE, while after anonymity is called Anonymity Error and denoted by AE. The BE classification error is 0.18 for Adult data, and 0.15 for Seer data. Figures 7 and 8 show the classification error comparison between the three anonymization methods. The comparison considered $k = 15$ in all trials. Three different data sizes were compared in the three anonymization methods.

Both datasets show similar results regarding AE value. The anonymization was conducted based on splitting data in BUG and MDTDS, while data was processed as a whole set in MDSBA. Table 5 illustrates the data split for 4.6 GB. In our trails, the first data size was small enough to avoid data split. Hence, the classification error was low. The low classification error indicates the high level of information gain. However, since data size increases, the anonymization algorithms need to split large data size into small blocks. This split affects the data equivalency to a certain extent. It was also noticed that

Table 5 Dataset of 4.6 GB split for both methods

K	4	8	12	16	20	50	100	170	250	300	400
MDTDS	4.6 GB	4.6 GB	4.6 GB	1.15 GB	650 MB	420 MB	230 MB	164 MB	110 MB	73 MB	38 MB
BUG	4.6 GB	4.6 GB	4.6 GB	2.3 GB	1.15 GB	770 MB	570 MB	350 MB	230 MB	170 MB	95 MB

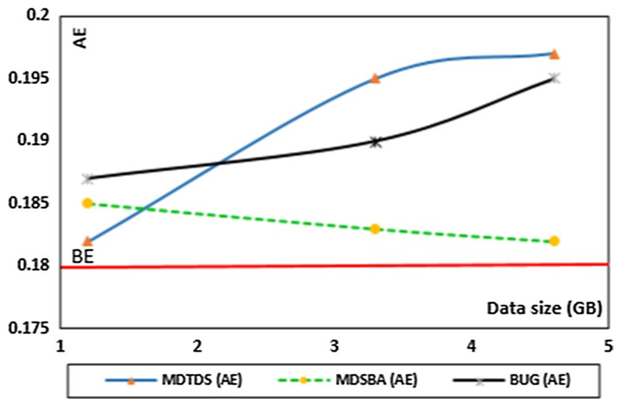

Fig. 7 Disruption comparison for $\overline{k} = 4$ and the three data. We developed an equation to find a benchmark for the level of information loss, or the anonymization impact. Named as Disruption. This diagram shows four different datasets; Adult, Seer cancer, Heart disease data and Kasandr. Each dataset was anonymized by Top-Down specialization, Bottom Up Generalization, and our MDSBA. It was found that our method was the lowest Disrupted

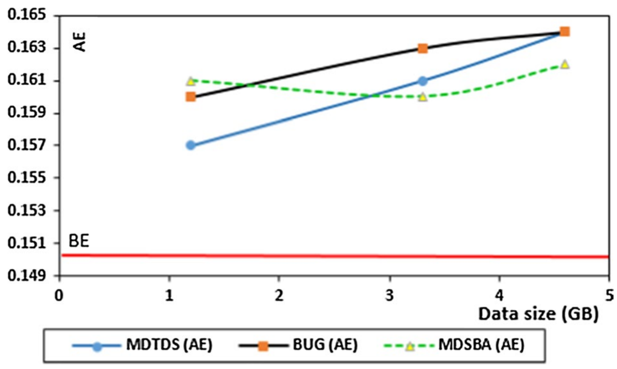

Fig. 8 Disruption comparison for $\overline{k} = 50$ and the three data. We developed an equation to find a benchmark for the level of information loss, or the anonymization impact. Named as Disruption. This diagram shows four different datasets; Adult, Seer cancer, Heart disease data and Kasandr. Each dataset was anonymized by Top-Down specialization, Bottom Up Generalization, and our MDSBA. It was found that our method was the lowest Disrupted

BUG and MDTDS perform better in smaller data size. This is because of their own algorithm's nature of keeping iteration and splitting until no further split is possible. In both trials, the value of AE showed a slight decline in MDSBA, as a reason of the equivalency increase parallel with the data size increase.

The first experiment does not accurately measure the amount of disruption. The output results rely on the classifier accuracy and its efficiency with such a data type. Our second experiment recalled our naïve disruption equation instead of measuring the prediction error percentage. The four datasets are used with 4.6 GB size of each. The anonymization methods are implemented by Bottom-Up Generalization, Top-Down Specialization, and MDSBA. Equations 8 and 9 are used to calculate $\mathfrak{D}$. The first part of the experiment aimed to measure the disruption values for the smaller values of $k = 4$, while the second part aimed to measure the disruption values for the larger values of k = 50. Figure 9 shows the results of the first part of the experiment, which indicates a minor contrast between the three methods. MDSBA shows a higher disruption than the others. However, the difference being around 0.0004 is very small and does not have any real impact on data analytics. Figure 10, shows the results of the second part of the experiment, which indicates a significant contrast between the three methods. MDSBA shows the lowest disruption level when k = 50.

In the second experiment, we aimed to find the anonymization level in MDSBA on the variation of $\overline{k}$. Figure 11 shows a gradual increase in $\mathfrak{D}$ on $\overline{k} \geq 50$ for all dataset. This $\mathfrak{D}$ increase is expected as a reason of the equivalency decrease with the large number of $\overline{k}$. This experiment was conducted with a data size of 4.6 GB for each dataset. The $\overline{k}$ increase may reduce the privacy violation risk, but however, it degrades the information usefulness. Hence, a tradeoff between security and disruption should be considered on assigning the values of k. From our previous and current experiments, we applied

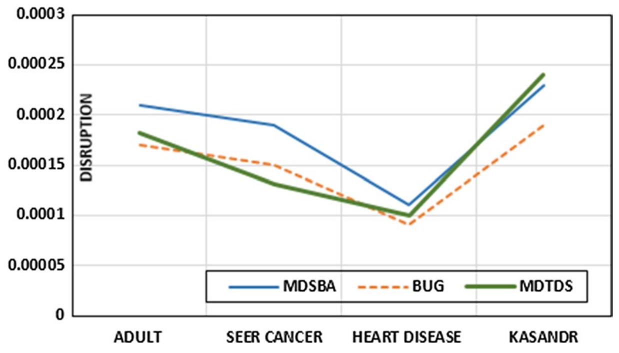

Fig. 9 The relationship between $\mathfrak{D}$ and $\overline{k}$ variation in MDSBA. Similar to Fig. 6. The Disruption value has increased parallel with the increased value of $\overline{k}$

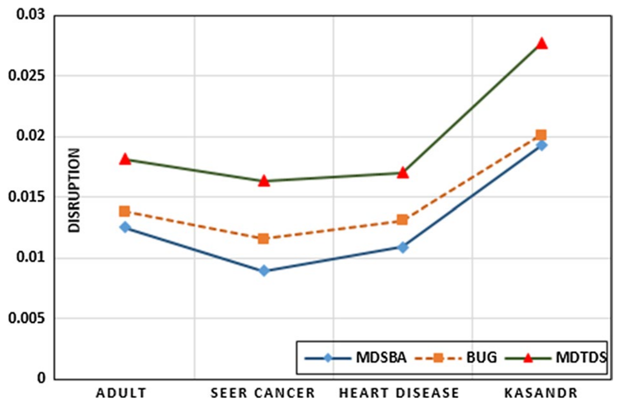

Fig. 10 Comparison between MDSBA and MDTDS with $\overline{k}$ and data size variations. This diagram illustrates our MDSBA performance, in comparison with the To-Down specialization method. MDSBA processing time is a prominent in small values of $\overline{k}$. However, MDTDS has lower latency on larger values of k, while MDSBA consumed the same time in both values of $\overline{k}$, when $\overline{k} = 50$, or $\overline{k} = 250$. This fixed timestamp refers to the time consumed on applying the *Obvious Guess*. Larger values of $\overline{k}$ requires less time spent on *Obvious Guess* processing, and more time on anonymization processing, and vice versa. Hence, the total of both processing times remains close similar for all $\overline{k}$ values

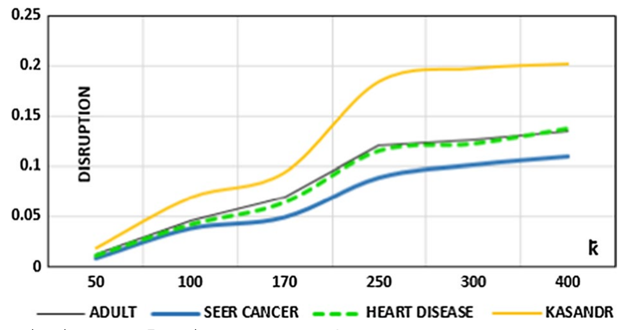

Fig. 11 The relationship between $\mathfrak{D}$ and variation in MDSBA

anonymization on verities of datasets, and we found that anonymizing most datasets will output a $\mathfrak{D}$ below 5%. However, this finding is a rough estimate and cannot be generalized to all datasets. Data $\mathfrak{D}$ is related to other factors, such as the data cumulative distribution, and the attributes probabilities, as described before. Also, low probability attributes are prone to a higher level of anonymization.

In the last experiment, we compared the processing time of MDSBA and MDTDS in Seer Cancer dataset, with the variations of $\overline{k}$ and data size. The value of $\overline{k}$ was replaced with k in MDTDS case, since MDTDS does not implement $\overline{k}$. MDTDS data was split into small size of data files, as shown in Table 5. As illustrated in Fig. 12, MDSBA processing time is a prominent in small values of $\overline{k}$. However, MDTDS has lower latency on larger values of k, while MDSBA consumed the same time in both values of $\overline{k}$, when $\overline{k} = 50$, or $\overline{k} = 250$. This fixed timestamp refers to the time consumed on applying the *Obvious Guess*. Larger values of $\overline{k}$ requires less time spent on *Obvious Guess* processing, and more time on anonymization processing, and vice versa. Hence, the total of both processing times remains close similar for all $\overline{k}$ values.

The three experiments showed promising results in performance and security regarding MDSBA. Increasing k value lowered the disruption results. This is essential in big data security, which depends on a higher equivalency number. MDSBA cannot accept smaller values of k, because its core structure relies on the gradual value of $\overline{k}$ value. This means that the minimum accepted k value cannot be smaller than k = 20. The reason for that is the minimum value of $\overline{k}$ is 2, and if a user was given 10% of k,

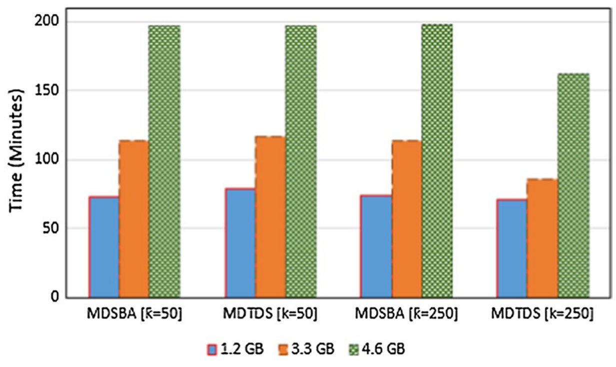

Fig. 12 Comparison between MDSBA and MDTDS with and data size variations

then $\overline{k} = 20 \times 0.1 = 2$ [18]. However, more experiments are needed to prove MDSBA's performance and security. Previous experiments were conducted on MDSBA and compared with several BUG and MDTDS methods. The experiments showed similar results in information gain results and performance improvement [26].

Conclusions

Big data analytics require proper anonymization tools and methods that furnish for secure procurement of information. Current anonymization methods may not be able to process large data sizes efficiently. Some of the anonymization methods are not capable of discriminating between the roles and potentially different access privilege levels of the users, resulting in the same levels of the information obscurity for their clients. To overcome some of these limitations, in this work, we have proposed a novel sensitivity-based on anonymization framework, MDSBA. The framework provides capabilities for efficient anonymization of big data, by incorporating contemporary anonymization tools, such as Pig Latin and UDF programming. Pig Latin shell behaves similar to SQL structure and can speed up the MapReduce operations, by grouping, filtering and splitting of data. Our proposed method can provide an output with various anonymization levels for users with multi-level of access rights. By adopting a bottom-up approach, our anonymization method reduces the rate of the anonymization loss. We have also made simple analytical comparisons of the obscurity levels of data provided by MDSBA, BUG, and TDS. The experiments, with three different datasets, have confirmed that information loss in MDSBA is 1.8 times lower than in MDTDS, and 0.5 times lower than in BUG, with the large values of k. The experiments have also shown the control of the degree of anonymity and data obscurity levels can be easily achieved by managing the MDSBA parameters. MDSBA, also, has shown a low time latency on anonymization process with low values of $\overline{k}$. Our future works will expand our experiments to include more sophisticated and complex settings to fine-tune the proposed approach. We also aim to establish a complete MDSBA framework, by providing supporting procedures that enhance the fine-grained access control between Federation Services and data Service providers. Moreover, we need to evaluate MDSBA performance in data stream model, since all current studies focus on data batch only. This may require continuous operations to apply anonymity on newly updated data records. Also, our gauge will evaluate MDSBA in various database frameworks, such as Spark and Storm. In addition, further studies are required to find the optimal values of k and $\overline{k}$.

Abbreviations

TDS: top-down specialization; BUG: bottom-up generalization; Q-ID: quasi identifier; MDSBA: multi-dimensional sensitivity-based anonymization; HDFS: hadoop distributed file system; UDF: user-defined function; $\overline{k}$: ownership level; ω: sensitivity factor; ψ: sensitivity level; Ø: obsolescence value; τ: aging factor; ρ: aging participation percentage; *SG*: fully-equivalent group; *SSG*: semi-equivalent group; *NG*: non-equivalent group; 𝔇: disruption.

Authors' contributions

MAZ wrote the main manuscript and the diagrams. CR participated in the set-up of the experiments and output results. SS reviewed and amended the manuscript scientifically, grammatically, and general text structure. All authors read and approved the final manuscript.

Acknowledgements

This research was conducted in Western Sydney University Labs. We are grateful for the technical support and establishment of the Hadoop domain. Special thanks are due to Mr. Guang Hui Deng.

Competing interests
The authors declare that they have no competing interests.

Availability of data and materials
Our experiments utilize three different standard datasets. They are: Adults data, taken from UCI machine learning ftp://ftp.ics.uci.edu/pub/machine-learning-databases/adult/adult.data. Seer Cancer Data, data is taken from SEERSTAT software, which is available on https://seer.cancer.gov/seerstat/software/. Kasandr data, taken from UCI machine learning https://archive.ics.uci.edu/ml/datasets/KASANDR. Heart Disease Data, taken from UCI machine learning available at https://archive.ics.uci.edu/ml/datasets/Heart+Disease.

Consent for publication
Not applicable.

Ethics approval and consent to participate
Not applicable.

Funding
The research works reported in this paper are part of the first author's Ph.D. degree pursued at Western Sydney University. The second and third authors are the supervisors of those works. The Australian government partially covers the funding for that degree under the Research Training Scheme (RTS).

Appendix 1. Pig Latin Script for Seer Data

```
REGISTER /usr/hadoop/trial.jar
--STAGE 1
Data = LOAD ‘hdfs://NameNode:9000/input/Seer.csv’ using PigStorage(',') as
(Y_O_D:int,COUNTY:int,RACE:chararray,AGE:int, DIAGNOSTIC:chararray);
G0 = FILTER Data by (DIAGNOSTIC matches ‘no positive histology’);
G1 = FILTER Data by (DIAGNOSTIC matches ‘Positive histology);
G2 = FILTER Data by (DIAGNOSTIC matches ‘Positive microscopic confirm);
G3 = FILTER Data by (DIAGNOSTIC matches ‘Positive laboratory test);
------------------------------------------------------------------------------------------------------------------------

-- [repeat for G1, G2, G3]
Data0= GROUP G0 by (Y_O_D, COUNTY, RACE, AGE)
PG0= foreach data0 generate count (G0) as cnt0:long, G0.Y_O_D as YOD, G0.COUNTY as COUNTY, G0.RACE as
RACE, G0.AGE as AGE, G0. DIAGNOSTIC as DIAGNOSTIC;
SG_0_1= FILTER PG0 by (cnt0 >=250);
SG_0_1_S = FOREACH  SG_0_1  GENERATE trial.ADJUST(Y_O_D,COUNTY, RACE, AGE, DIAGNOSTIC);
STORE SG_0_1_S into 'hdfs:// NameNode:9000/mdsba/output/RESULTS/SG_0_1_S using PigStorage(',');
SSG_0_S= FILTER PG2 by (cnt0 <250);
SSG0= FOREACH SSG_0 GENERATE trial.ADJUST(Y_O_D,COUNTY, RACE, AGE, DIAGNOSTIC);
STORE SSG0 into 'hdfs:// NameNode:9000/mdsba/input/G0/SSG_0' using PigStorage(',');
--STAGE 2 [repeat stage 2 for G1, G2, G3]
data1= LOAD 'hdfs:// NameNode:9000/mdsba/input/G0/SSG_0' using PigStorage(',') as
(Y_O_D:int,COUNTY:int,RACE:chararray,AGE:int, DIAGNOSTIC:chararray);
STAGE_2= GROUP data1 by (Y_O_D,COUNTRY,RACE);
PG1= foreach STAGE_2 GENERATE cnt1 (data1) as cnt1:long, data1.Y_O_D as YOD, data1.COUNTY as
COUNTY, data1.RACE as RACE, G0.AGE as AGE, G0. DIAGNOSTIC as DIAGNOSTIC;
SG_0_2= FILTER PG1 by (cnt1 >=250);
SG_0_2_S = FOREACH L_SG_0_2 GENERATE trial.SSG_P1(Y_O_D,COUNTY, RACE, AGE, DIAGNOSTIC);
STORE SG_0_2_S  into ‘hdfs:// NameNode:9000/mdsba/output/RESULTS/ SG_0_2_S using PigStorage(‘,’);
L_SSG_1= FILTER PG1 by (cnt1 < 250);
SSG_1 = FOREACH  L_SSG_1  GENERATE trial.ADJUST(Y_O_D,COUNTY, RACE, AGE, DIAGNOSTIC);
STORE SSG_1 into ‘hdfs:// NameNode:9000/mdsba/input/G0/SSG_1 using PigStorage(‘,’);
--STAGE 3 [repeat stage 3 for G1, G2, G3]
data2= LOAD ‘hdfs:// NameNode:9000/mdsba/input/G0/SSG_1/’ using PigStorage(‘,’) as
(Y_O_D:int,COUNTY:int,RACE:chararray,AGE:int, DIAGNOSTIC:chararray);
STAGE_3= group  data2 by (RACE);
PNG1= foreach STAGE_3 generate cnt2 (data2) as cnt2:long, data2.Y_O_D as YOD, data2.COUNTY as COUNTY,
data2.RACE as RACE, data2.AGE as AGE, data2.DIAGNOSTIC as DIAGNOSTIC;
L_NG_0= FILTER PNG1 by (cnt2 >=250);
NG_0 = FOREACH L_NG_0  GENERATE trial.SSG_P2(Y_O_D,COUNTY, RACE, AGE, DIAGNOSTIC);
STORE NG_0 into ‘hdfs:// NameNode:9000/mdsba/output/RESULTS/SG_0_3 using PigStorage(‘,’);
L_NG_1= FILTER PNG1 by (cnt2 < 250);
NG_1 = FOREACH  L_NG_1  GENERATE trial. ADJUST _P(Y_O_D,COUNTY, RACE, AGE, DIAGNOSTIC);
STORE NG_1 into ‘hdfs:// NameNode:9000/mdsba/output/RESULTS/SG_0_4 using PigStorage(‘,’);
```

Publisher's Note

Received: 24 August 2017 Accepted: 17 November 2017
Published online: 08 December 2017

References

1. Sweeney L. Anonymity: a model for protecting privacy. Int J Uncertain Fuzziness Knowl-Based Syst. 2002;10(05):557–70. https://doi.org/10.1142/S0218488502001648.
2. Wang K, Yu PS, Chakraborty S. Bottom-up generalization: a data mining solution to privacy protection. USA; 2004. p. 249–56.
3. Fung BCM, Wang K, Yu PS. Top-down specialization for information and privacy preservation. USA; 2005. p. 205–16.
4. Irudayasamy A, Arockiam L. Parallel bottom-up generalization approach for data anonymization using map reduce for security of data in public cloud. Indian J Sci Technol. 2015;8(22):1.
5. Zhang X, Liu C, Yang C, Chen J, Nepal S, Dou W. A hybrid approach for scalable sub-tree anonymization over big data using MapReduce on cloud. J Comput Syst. 2014;80:1008–20.
6. Zhang X, Yang LT, 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(2):363–73.
7. Sun Y, Yin L, Liu L, Xin S. Toward inference attacks for k-anonymity. Pers Ubiquit Comput. 2014;18(8):1871–80.
8. Zhang X, Yang C, Nepal S, Liu C, Dou W, Chen J. A MapReduce based approach of scalable multidimensional anonymization for big data privacy preservation on cloud; 2013. p. 105–12.
9. Shtern M, Simmons B, Smit M, Litoiu M. Toward an ecosystem for precision sharing of segmented Big Data. In: Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on. IEEE; 2013. p. 335–42.
10. Wu X, Zhu X, Wu GQ, Ding W. Data mining with big data. IEEE Trans Knowl Data Eng. 2014;26(1):97–107.
11. Kwon Y. Hadoop internals. Lecture notes from Washington University; 2014. https://courses.cs.washington.edu/courses/cse344/11sp/lectures/cse344-guest-yckwon.pdf. Accessed 10 Feb 2017.
12. Project H. HDFS users guide. Apache. https://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html (2016). Accessed 2016.
13. Dokeroglu T, Ozal S, Bayir MA, Cinar MS, Cosar A. Improving the performance of Hadoop Hive by sharing scan and computation tasks. J Cloud Comput. 2014;3(1):1.
14. Gold S. Python: Python programming learn Python programming in a day-a comprehensive introduction to the basics of Python & computer programming. ACM Digital Library; 2016. https://dl.acm.org/citation.cfm?id=3055663. Accessed 20 Apr 2017.
15. Grimmer M. 'High-performance language interoperability in multi-language runtimes. In: Proceedings of the companion publication of the 2014 ACM SIGPLAN conference on systems, programming, and applications: software for humanity. ACM; 2014. p. 17–9.
16. Davis AL. Modern programming made easy: using Java, Scala, Groovy, and JavaScript. Berkeley: Apress; 2016.
17. Lublinsky B. Professional hadoop solutions. Hoboken: Wiley; 2013.
18. Al-Zobbi M, Shahrestani S, Ruan C. Implementing a framework for big data anonymity and analytics access control. In: Trustcom/BigDataSE/ICESS, 2017 IEEE. IEEE; 2017. p. 873–80.
19. Al-Zobbi M, Shahrestani S, Ruan C. Sensitivity-based anonymization of big data. In: Local computer networks workshops (LCN workshops), 2016 IEEE 41st Conference on. IEEE; 2016. p. 58–64.
20. Harrington JL. SQL clearly explained. Burlington: Morgan Kaufmann; 2003.
21. Pasupuleti P. Pig design patterns. Birmingham: Packt Publishing; 2014.
22. Li K, Reichenbach C, Smaragdakis Y, Diao Y, Csallner C. SEDGE: symbolic example data generation for dataflow programs. In: Automated software engineering (ASE), 2013 IEEE/ACM 28th International Conference on. IEEE; 2013. p. 235–45.
23. Govindaraju V. Big data analytics. Oxford: Elsevier Science; 2015.
24. Fung BCM, Wang K, Philip SY. Anonymizing classification data for privacy preservation. IEEE Trans Knowl Data Eng. 2007. https://doi.org/10.1109/TKDE.2007.1015.
25. Guller M. Big data analytics with spark a practitioner's guide to using spark for large scale data analysis. Berkeley: Apress; 2015.
26. Al-Zobbi M, Shahrestani S, Ruan C. Multi-dimensional sensitivity-based anonymization method for big data. In: Elkhodr M, Shahrestani S, Hassan Q, editors. Networks of the future: architectures, technologies, and implementations. Boca Raton: Chapman and Hall/CRC Computer and Information Science Series, Taylor & Francis; 2017. p. 448.