# CHAPTER 4 – implementing Sensitivity-Based Anonymization method by Hadoop Ecosystems

MapReduce transaction method is different from the classical transaction procedure in analytics process. MapReduce divides data process into two main tasks; reading data from multi-repositories and aggregating results in a reduced output. This imposes a new method of disposition in privacy-related operations. The anonymization process should be amended to fit the reading, shuffling and reducing of data, as per MapReduce environment. Some privacy preservation methods are modified to fit the MapReduce framework and perform parallel data-intensive computations on commodity computers [1]. Computation reads input data from a distributed file system, which splits the data into multiple chunks. Each chunk is assigned to a mapper which reads the data, performs some computation, and emits a list of key/value pairs. In the next phase, Reduce phase combines the values belonging to each distinct key according to some functions and writes the result to an output file. The framework ensures fault-tolerant execution of mappers and reducers while scheduling them in parallel on any node in the system [2].

Since the MapReduce operations include; split, Map, shuffle and reduce, therefore, any practical security solution should consider these transactions. Any tweaking in the available algorithms should consider the milestones of the scale-up efficiency and the data privacy [3]. Recently developed methods in k-anonymity moved toward finding parallelization techniques in the anonymization algorithms. The techniques should be able to split the massive size of data into smaller blocks, so the algorithms overcome the intensive and recursive computation operations. Several methods have split data for parallelization such as; Two-phase Multi-Dimensional Top-Down Specialization method TPMDTDS [4] and the method in [5].

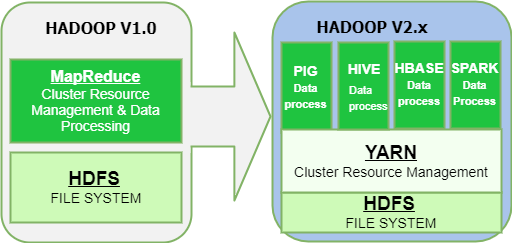
The parallelization methods are proposed during the early release of Hadoop. Currently, MapReduce can be easily implemented by using Pig Latin, Hive, or Spark, which makes the MapReduce job easier. This concern recalls for an indirect method that can provide better-performed operations. Previously, Hadoop scripts are implemented by programming languages only; such as Java. Currently, Java can be replaced by ecosystems like, Pig Latin queries or Hive. However, Java use can be reduced to the minimal, and on need only.

Before implementing a proper parallelization algorithm, MapReduce security in operating big data analytics should be investigated. As mentioned in the previous chapters; security is one of the main concerns in building any anonymization framework. The security concerns conclude three main attacks of side link attack: state attack, privacy attack, and time attack. The main security threats fall between state attacks and privacy attacks. The state attack security includes data and queries. User’s queries, analytics results, and the analytics process must be protected. The privacy attack includes all kinds of data re-identification. The next few sections describe state security and protection through MapReduce framework. Since MapReduce is the dominant framework for MDSBA, then the security and protection methods should be partially related to Hadoop security. Securing Hadoop is the safeguard for analytics protection.

## Hadoop in Data Analytics

Hadoop is a MapReduce framework that is able to process a large size of data. Many open source frameworks are recently developed for parallel distributed processing. The massive growth of data urged developers to produce more efficient computation operations and frameworks. However, Hadoop MapReduce gained popularity for its flexible core design. Since Hadoop version 2, one major Hadoop component is added to deal with several new developed frameworks, known by Yet Another Resource Negotiator (YARN). As shown in Figure 4.1, in Hadoop new versions 2 and 3, YARN took over the cluster management from MapReduce. The data management is handled to several ecosystems such as; Sqoop, Flume, Pig, Hive, Spark, Storm and many others. Actually, the principle of Hadoop is dramatically changed since version 2. This jump of the core design recalled a need for changing the analytics techniques. The majority of the previously developed data analytic and anonymization methods were related to the first version of Hadoop [6].

The new versions of Hadoop have provided a set of ecosystems to facilitate the data analytics tasks. Hadoop ecosystems operate with tools reside at the top of Hadoop. These tools are essential for big data management and operations. The tools may be protected by applying security features on Hadoop core. Security features of Hadoop consist of Authentication, Service Level of Authorization, Authentication for Web Console, and Data Confidentiality. Hadoop core structure comprises three main parts that should be protected; HDFS, YARN, and JobHistory. The three parts are secured by using Hadoop secure mode, so each user and service needs to be authenticated by Kerberos before using Hadoop services [7].



*Figure 4.1- Comparison between Hadoop v1 and v2.*

As mentioned earlier, Hadoop security is able to protect data, user’s access, and processes. The three services can be protected by Hadoop secure mode. It can be said that Hadoop secure mode protects against state attack. However, the secure mode is unable to protect data against privacy attack. The privacy attack appears on the data level by the authorized users, who should not be able to explore sensitive data. Private and sensitive data cannot be available to every authorized user. Instead, limited users should be eligible to access such sensitive data. Authorized user means a user that is permitted to access all or part of the data.

Big data is correlated to a large number of users, which in turn increases the percentage of exploring prohibited attributes. There is a need for a framework that can organize and control the amount of data accessed, by implementing access control methods. Hadoop security features are unable to provide such a level of data protection. Hence, Hadoop security features secure data from any unauthorized access only and without providing a robust access control.

### Hadoop Core

Hadoop version 1 implements JobTracker daemon service for submitting and tracking MapReduce jobs. In version 2, this was replaced by a Resource Manager (YARN). The resource manager, located in NameNode, communicates regularly with the Node Manager, located in DataNode. The Node Manager provides the Resource Manager continuously with the available resources from disks, CPU, and RAM. If a client launched a job, the Application Master, located in the NameNode, will create a list of Map Tasks and requests containers from the Resource Manager to execute the tasks. The Resource Manager responds by creating a number of containers on the available nodes. The number of containers depends on the available resources in each node of the cluster. Each container is a single JVM that is mapped to one or more tasks [8].

The number of created tasks depends on the number data blocks. For instance, if a data size was 2 TB, and each data block was setup to 128 MB, then the number of scheduled tasks will be around 16,400 tasks. This massive number of tasks is executed on a limited number of containers. The mapper reads several blocks from the HDFS and executes. The process continues until the end of the data blocks. The reducer may start shuffling data while the mapper is still running. The Resource Manager assigns nodes for tasks based on data locality principle. Hence, it executes the mapper on the nodes where the block resides. The locality principle is possible by data replication created by HDFS. In map or reduce phase, YARN may spill some data blocks to the disk, if the memory resources were almost full [9].

In User-Defined Function (UDF), all registered operations are executed in a black-box [10]. A separate JVM is created beyond the Resource Manager scope. YARN does not manage this JVM, and it cannot be considered as a node container. The created JVM is controlled and managed directly by the operating system. YARN registers the UDF file, for instance JAR, transfers the data to the created JVM, and waits for the output. UDF may cause two major problems if used with a massive data size. The first problem is the non-parallelization of the tasks, which creates a bottleneck for the operation performance. The second problem is the lack of spilling data on the large data size, which leads to a Java heap memory failure. Hence, if the flowing data size to the UDF was large, then the unexpected process termination is high. For these reasons, the implemented UDF should be carefully studied, in order to mitigate the large data flow impact.

### Hadoop Ecosystems

The phrase of (Hadoop ecosystems) has been generalized to conclude any recently developed frameworks that operate at the top of Hadoop. For instance, Spark, and Storm are known by Hadoop ecosystems. Most developed big data frameworks implement one of Hadoop components for data processing. The new Hadoop versions, two or version three, can accommodate several frameworks by keeping YARN engine at the top of the nodes management. YARN is able to process multiple tasks at the same time. YARN boosts Hadoop to incumbent any new technologies found within the data center, so they can take advantages of cost-effective, linear-scale storage and processing [11].

The new Hadoop ecosystems are able to process interactive, stream, and real-time operations. The interactive is presented by batch data stored in files or database. The real-time and stream are closely similar in operational concept. The major difference between them is the processing time spent in completing each one. The processing time of data streaming should not exceed milliseconds, while the real-time may take longer time, around two to three seconds. The MDSBA core structure aims to prepare data for analyzers. The preparation implies data anonymization as requested by data owners. Data analyzers are then ready to access the anonymised data for statistical analytics. Datasets can be archived or live data, hence, anonymizing archived data technique is different from live data. Next chapters, will be focused on archived data, while the live data will be discussed in chapter 7 [12].

Apache Hadoop ecosystems were developed with Java Virtual Machine (JVM) core. The JVM is the major tool of creating nodes, workers, and multiple processes. Ecosystems were shipped with scripting languages that help developers controlling data flow. The scripts provide the SQL-like commands to enable a better management over big data. Hive, for example, adopts a scripting language, called Hive Query language or HiveQL. The script is able to create, alter select, insert, and describe database tables from Hive data warehouse. Hive supports analysis of large datasets stored in [HDFS](https://en.wikipedia.org/wiki/HDFS) and other compatible file systems such as S3. Pig also implements Pig Latin scripting language. The script provides another SQL-like scripting language that can manage large data. The script can read, group, filter, and join multiple datasets stored in HDFS or in any other database management systems. However, the SQL-like scripts are unable to provide complete solutions for developers, Therefore, User Defined Functions (UDF) can be embedded in the script to provide powerful programming algorithms [13].

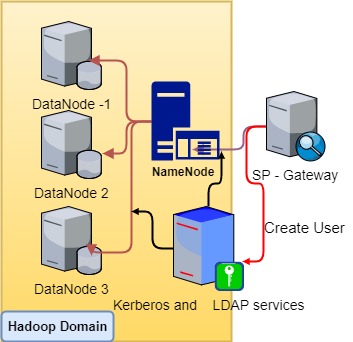
## Hadoop security

By default, Hadoop is shipped with no security mode. The user may access illegal files and directories, especially that Hadoop does not provide any authentication service. In the non-secure mode, users can be created in NameNode servers only, while the DataNodes servers may not contain the specific username. However, YARN is able to access all DataNodes with the authorization of NameNode user. In the secure mode, the user must be added to all DataNodes, which is hard to implement in a multi-user environment. However, this can be implemented by creating an LDAP or NIS domain to conclude NameNodes and all related DataNodes in one domain. LDAP provides a practical method for the Single Sign-On (SSO) over the local network. LDAP and Kerberos are proper combinations to secure the user’s access to services, files and folders. Kerberos provides the authorization level, while LDAP provides the authentication level. If LDAP was implemented without Kerberos, then a security concern gives users privileges to access any data block stored in DataNode. On the other hand, implementing Kerberos without LDAP oblige administrators to add the user to every DataNode, which is impractical. Hence, both services of LDAP and Kerberos should be implemented together.

### Implementing Hadoop in LDAP Domain

Lightweight Directory Access Protocol (LDAP) domain is an essential part of Hadoop security mode. Kerberos bears with the domain users, who are created merely in (LDAP), so the created keys are mapped to domain users by assigning Security Identifier to each user. Hadoop NameNodes and DataNodes should be available in one Name Service or domain. In UNIX, the Name Service can be established by Network Inform Service (NIS), LDAP, DNS or BIND [14]. This is essential in distributed environments for a distributed authorization and authentication through a centralized single sign node. Users need to access the domain NameNode with a single sign, hence the rest of the DataNodes in the cluster should inherit the authorization and authentication details. Newly created users are merely added to the centralized LDAP server to gain the authorization and authentication access. LDAP with Kerberos are highly recommended for their compatibility in protecting network objects [15].

MDSBA framework implements both Kerberos and LDAP services to provide a level of authorization and authentication. Figure 4.2 illustrates Hadoop domain structure in MDSBA. The service provider establishes a gateway between user’s access point and Hadoop domain. A centralized server contains LDAP service to connect all NameNodes and DataNodes of Hadoop cluster. New users must be added to LDAP server before being authorized to access NameNode. The authorization includes; some UNIX processes, Hadoop processes, local directors, and specified directories in HDFS.



*Figure 4.2- Kerberos and LDAP server in MDSBA.*

### Applying Kerberos for Hadoop Secure Mode

One of the essential Hadoop security features is enabling a proper authentication to Hadoop services. This can be implemented by providing Kerberos authentication service. Kerberos requires a Key Distribution Centre sever (KDC), which provides the entire network with Authentication Service (AS) and Ticket Granting Service (TGS). Kerberos provides a powerful tool to protect the cores of Hadoop structure; HDFS, YARN, and MapReduce. The protection can be established by various internal secret keys; Delegation Token, Block Access Token, and Job Token [16].

It is essential to generally understand the three internal secret keys. The Delegation Token is the first secret key given by Kerberos to the user. Once the user attempts to access the NameNode, then he/she will be requested to authenticate before given the Delegation Token. The second token is the Block Access, which is merely implemented for HDFS. Any Hadoop client requesting data from HDFS needs to fetch the data blocks directly from DataNode after reading the block ID from NameNode. The Block Access Token secures the transferring of user privileges to DataNode. The main purpose of the Block Access Token is to ensure that only authorized users are able to access the data blocks stored in DataNodes. When a client needs to access the data stored in HDFS, it requests NameNode to provide the block IDs for the files. NameNode verifies the requested user's permissions for the file and provides the list of block IDs and DataNode locations. The third shared secret key is the Job Token key, which is stored locally with the user profile, and used by the Task Tracker to protect the user’s submitted task. Thus, the Job Token ensures that the user who submits a job has only access to the authorized local file systems of task nodes [17].

Kerberos protects the three entire Hadoop cores, and the related ecosystems. The three entire cores include YARN, HDFS, and MapReduce (mapred), while the ecosystem tools include Sqoop, Pig, Hive, Oozie, Spark, Flume, Hbase, and others. YARN is responsible for Resource Manager, which monitors and manages Node Manager and Application Master. YARN, in Hadoop v2 and higher, manages the infrastructure of Hadoop environment, so it acts as an operating system for Hadoop. HDFS is the file system that is responsible for file naming and data blocks storing and replicating between nodes. The third part is mapred, which manages MapReduce and JobHistory server. The security mode is initiated by creating Kerberos principals in the KDC database. Kerberos then creates a key tab file containing pairs of Kerberos principals and encrypted keys derived from Kerberos user password. One key tab is created for each one of the three main services, HDFS, YARN, and mapred. Once key tabs are created, they need to be moved to Hadoop configuration folder such as /usr/hadoop/etc/hadoop/ in CentOS 7. This folder must be accessible for all Kerberos users. Moreover, the configuration contains The XML configuration files of core-site.xml, hdfs-site.xml, and mapred-site.xml. These two configuration files are edited to map the key tabs locations. The hdfs-site.xml file contains two property names of (Hadoop.security.authentication, Hadoop.security.authorization) with values of (Kerberos, true) sequentially. The other two XML files should be modified accordingly [18].

Kerberos follows similar steps in authorizing all Hadoop cores and ecosystems. Kerberos bears with domain users, who are created merely in LDAP, so created keys are mapped to domain users by assigning Security Identifier to each user. NameNode and DataNodes servers must be members of one LDAP domain. To protect Hadoop core services and ecosystem, the administrator needs to create a principal for each process, and then a keytabs. The keytabs then need to be copied to the configuration folder first, and then mapping their locations by XML configuration files before starting Hadoop services. On the other side, users need to initiate their own Kerberos key by using ‘kinit’ command after authenticating throughout the LDAP domain services. Figure 4.3 describes the steps that Administrators should follow to secure any service by Kerberos. In Hadoop case, the steps include; creating service principals for YARN, HDFS, and mapred, and then creating keytabs for the three services, and then copying keytabs to Hadoop configuration folder, and updating Hadoop XML configuration files with the keytabs locations, and finally starting Hadoop [19].



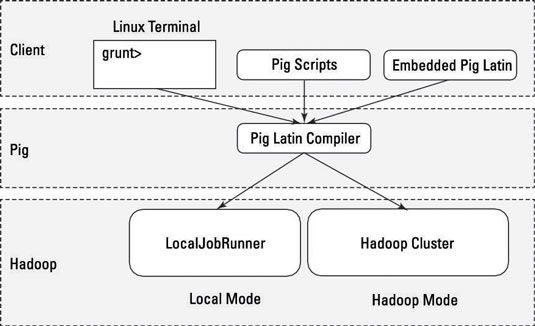
*Figure 4.3- Kerberos general steps to secure services by Administrators.*

## Deploying Sensitivity-Based Anonymization by using Hadoop Ecosystems

### Pig Latin Core Structure

MapReduce specifications and the available tools should be considered on working with Hadoop. Before delving into MDSBA technical details, we should be aware of the Hadoop programming tools, and its general structure. MDSBA is structured and implemented based on Hadoop ecosystem scripts such as Pig Latin and Spark. Pig Latin is a programming tool that provides a high-level platform for Apache Hadoop programs operations. Pig is able to execute scripts in three different modes, these are local, Tez, and MapReduce modes. The local mode is fast, reliable and suitable for developers. In local mode, all scripts are run on a single machine without requiring Hadoop MapReduce and HDFS. When running local mode, the Pig program runs in the context of a local Java Virtual Machine, and data access is accomplished via local file systems on a single machine. In MapReduce mode, Pig script is executed on the Hadoop cluster. In this case, the Pig Script gets converted into a series of MapReduce jobs that are then run on Hadoop [20].

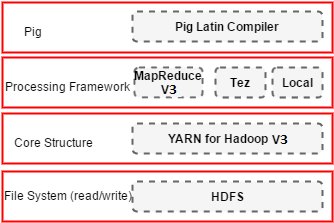
As shown in Figure 4.4, Pig compiler resides at the top of Hadoop. The script may run through Hadoop editor, known by a grunt, or by using UNIX terminal. The script is stored in a file extension of (.pig).Pig Latin mimics the programming structure in Java and SQL idioms and notations, which makes Pig Latin programming similar to that of SQL for RDBMSs [21]. Pig Latin can be extended using User Defined Functions (UDFs), where users can write Java, Python [22], JavaScript, Ruby [23] or Groovy [24], and then call directly from the pig script.



*Figure 4.4. Illustrates Pig Latin structure.*

In the following experiments in this chapter, Java is implemented as a UDF combined with Pig Latin script. The script reads the database by using Hadoop reading process. The process splits data into large blocks of 128 MB, and each block is mapped to a DataNode. The framework ensures fault-tolerant execution of mappers and reducers while scheduling them in parallel by using the Job scheduler that is available in NameNode [2]. MapReduce evolves the use of parallel and distributed computing for a large size of data, which may exceed 10’s of Terabytes. Since the MapReduce uses split, map, shuffle and reduce, therefore, any practical security solution should take these main processes in the consideration [3].

MDSBA mimics the MapReduce operational steps. The data block size, 128MB, is large enough to accommodate the large size of big data. Hive and Pig ecosystems operate as Hadoop tools. Traditionally, SQL language is the database dominator to manage and alter data. 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 [15]. 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 very large data sets. Thereby, substantial parallelism and a slew of optimization techniques are supported. Pig provides customized support for the UDF, by supporting many common languages such as Java. Similar to Hive, Pig supports ad-hoc queries, joins, and other SQL-like operations [19]. Pig Latin is a combination notation of SQL-like and Java idiom. However, Pig cannot implement high-level programming notations, therefore, UDF is essential. [20].



*Figure 4.5-Pig structure at the top of the processing framework*

MDSBA is proposed mainly for the MapReduce structure. MDSBA divides the anonymization into multi jobs including; reading, filtering, grouping, and filtering data again, to create SG and NG groups. The master server divides the user’s query into the multi-job process, and each job is divided into multi-tasks. The aim of MDSBA is reducing the processing time in each DataNode, and dividing tasks among the distributed system in the cluster.

### Anonymization by Pig Latin scripts

MDSBA was proposed to fit the new Hadoop ecosystems. One of Hadoop ecosystems is Pig, which is used to program MapReduce in an SQL-like script. Pig script is essential to anonymize data before permitting user’s access. The script is generated based on some XML files received from the federation service or single sign-on authority. The XML file contains the user’s access level, which consists of and ψ values. More details about the general structure of MDSBA is described in the next chapter. Pig algorithm relies on vertical and horizontal grouping on applying anonymization. The horizontal grouping is applied by dividing the data attributes into small Q-ID groups. That is each group G(QID) may comprise two to four Q-IDs. The groups are created before commencing the anonymization process. Each group is mapped to one or more of business role(s). On the other hand, vertical grouping is applied during the anonymization process to fulfil the k-anonymity conditions. The vertical grouping comprises of four different types of groups, G, SG, SSG, and NG. The grouping was explain in 3.2.3 section.

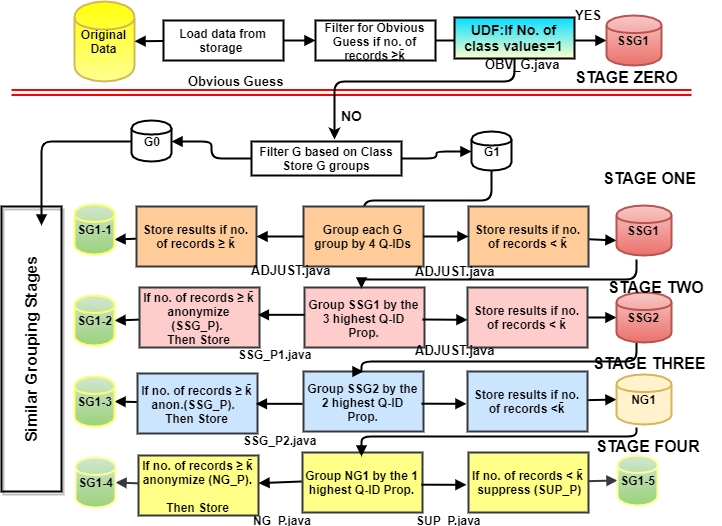
This section focuses on implementing vertical grouping by Pig Latin script. The vertical grouping is the core concept of data anonymization in MDSBA. Pig Latin algorithm can be divided into four major procedures: filtering, grouping, ungrouping, and masking. The first two procedures are accomplished by Pig Latin script, while the ungrouping and masking are accomplished by UDF. SQL-like script have the capabilities of grouping and filtering data records, but ungrouping and masking may require a high-level language to execute more complicated algorithms. The basic principle of anonymization is dividing anonymization process into stages. The number of stages varies depending on the number of G(QID). As described before, Q-ID group may contain a number of two to four Q-ID attributes. Hence, the number of generated scripts depends on the number of Q-ID attributes in the group. If the number of Q-ID groups is G(QID)=4, then the processing stages are also four. If the number of Q-ID groups is G(QID)=3, then the processing stages are three. Finally, if the number of Q-ID group is G(QID)=2, then the processing stages are 2. Figure 4.6 illustrates 4 stages for G(QID)=4, while Figure 4.7-A and Figure 4.7-B illustrate 3 and 2 stages for G(QID)=3, and G(QID)=2 consequently.

The figures show the stages starting from filtering each sensitive class value and storing it in a separate G group. Each G group is processed individually. The filter of the class value is essential to reduce the shuffling times in the reduce phase. It also supports the parallelization of large data size, by splitting data logically, and distributing loads among the cluster nodes. In each stage, the full-equivalent records are aggregated in an SG output. These records do not need any further masking or processes. In the first stage, Pig Latin script can achieve this aggregation of SG output, by following three methods, GROUP, COUNT, and FILTER. The GROUP clause in SQL-like is used to arrange identical attributes into groups, to conclude data bags. The COUNT clause follows the GROUP to count the number of aggregated records, which represents the number of equivalent records in each data bag. Finally, the FILTER is used in collaboration with a conditional statement to filter out the number of equivalent records greater than. The three commands syntax for three Q-IDs can be presented as:

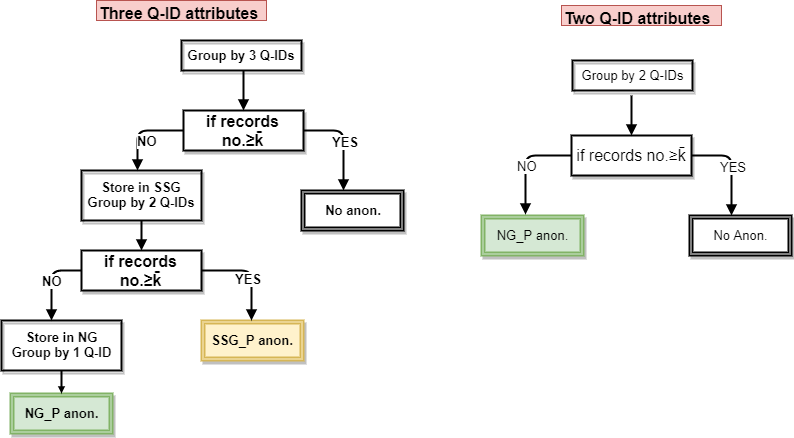
***G= GROUP data by (QID1, QID2, QID3);***

***SG= foreach G generate COUNT (data) as cnt: long, data.QID1 as QID1, data.QID2 as QID2, data.QID3 as QID3;***

***SG\_1= FILTER SG by (cnt >= );***



*Figure 4.6- Four Processing stages to anonymize four Q-IDs group.*



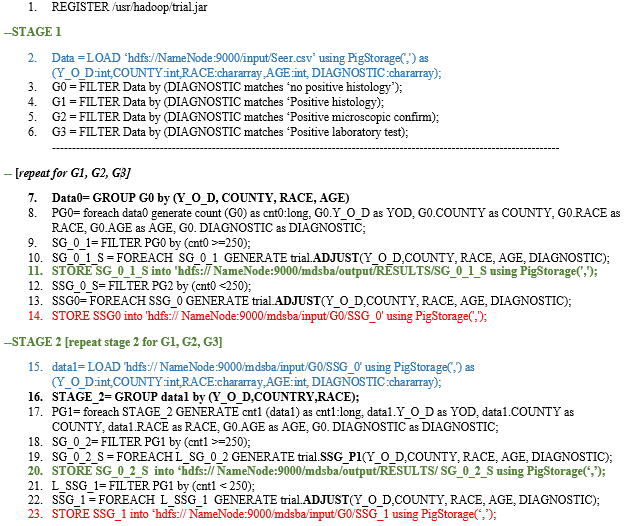
|  |  |
| --- | --- |
| *Figure 4.7-A. Anonymization process of three Q-IDs* | *Figure 4.7-B. Anonymization process for two Q-IDs* |

In the following stages, and after the first stage, the SSG groups cannot be converted to SG groups by following similar clauses of GROUP, COUNT, and FILTER; instead, the UDF program accomplishes this task. This is because the masking operation is needed for semi-equivalent records. In the first stage, masking is not required, so there is no need for UDF programs. However, the next stage requires masking operations for the semi-equivalent records. Therefore, the UDF program can apply masking before filtering out the equivalent records. The UDF program can complete both tasks of masking and aggregate the SG records.

Figure 4.6 is divided into four main stages. Each stage generates a set of data groups. Stage one filters or splits data based on class values to generate G groups. This is implemented by loading data from HDFS location, and before the split, so each class value is created in a separate G group. Each G group is then processed separately. After creating G groups, a GROUP 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 full-equivalent records and store them in SG location. The non-equivalent records are stored in a separate SSG1 location. In stage two, the largest three Q-ID probability values of SSG1 data will be grouped for semi-equivalency. The records that pass the semi-equivalency criteria are further anonymized. The anonymization is applied to the lowest probability value of Q-ID attributes and finally stored in SG location. The records that fail the semi- equivalency criteria are stored in 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 is 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[20]. 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 value, if they are larger, then an anonymization will take a place. Stage two filters three Q-ID attributes of SSG1 data, so if cnt ≥, 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 ≥, or by SUP\_P if <. A complete Pig script for Adult data is available in Appendix 1, with three Q-ID attributes.

The created SG groups of equivalent records are collected together under one HDFS directory. In Figure 4.6, all SG groups including, SG1-1, SG1-2…... SG1-5 are collected with the other SG groups from the rest of the G groups. Notice that groups of G, SSG, and NG are all temporary and they must be converted to SG by applying the masking processes. As mentioned earlier, the first stage groups and filters data with a need for UDF programs. Next stage, the UDF programs apply masking, aggregate SG records and ungroup the grouped records.

#### Pig Latin Script Example



*Figure 4.8- Pig Latin script example for the first two stages.*

Let us study the following complete example of anonymizing data by implementing Pig Latin script, as shown in Figure 4.8. The anonymization is applied to Seer Cancer Data. Table 4.2 shows the suggested Q-IDs, and the probability for each attribute and Table 4.3 shows the Class attributes with four sensitive values.

*Table 4.2- 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 4.3- Seer Cancer Data Class (four sensitive values).*

|  |
| --- |
| DIAGNOSTIC |
| no positive histology |
| Positive histology |
| Positive microscopic confirm |
| Positive laboratory test |

Suppose that data owner assigns k=400, and aging factor τ= -0.02. A user attempts to access this data with an ownership level =250. Referring to Table 4.2, and the sensitivity equations described in the previous chapter, the =0.33, = 2.5×10-06, so the sensitivity factor is found to be ω= 0.124. The data will be anonymized with a sensitivity level ψ=ω+ τ=0.124-0.02≈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, which is accepted value since it is smaller than ψ =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.

Referring to Figure 4.7, Pig Latin script controls the data flow by dividing the program into stages. Stage one filters the class attribute into four G groups, as shown in Table 4.3. The filtration commands are shown in Figure 4.7 / lines 2-6. Each G group is processed by grouping all Q-ID attributes **(Y\_O\_D, COUNTY, RACE, AGE)** in the first stage. Line 9 presents the filter with a conditional statement, where all cnt ≥ 250 are equivalent records, and therefore, they should be stored in the SG group after adjusting them, as shown in lines 10 and 11. The variable (cnt) counts the number of equivalent records as shown in line 8. The rest of the data records that cnt < 250 will be transferred to ADJUST program, and stored in SSG location, as described in lines 12 and 13. In stage 2, SSG data is retrieved, and grouped again, as shown in lines (15 – 16). In this stage, the grouping is conducted by three Q-ID attributes only **(Y\_O\_D, COUNTRY, RACE).** The attribute **(Age)** is excluded, since it is the lowest probability value among the four Q-ID attributes, as illustrated in Table 4.2.In this stage, the first anonymization is applied in line 19 by calling the program SSG\_P1, and transferring the grouped data to the program SSG\_P1. The rest of the data records that cnt < 250 will be transferred to ADJUST program, and stored in SSG location, as described in lines 22 and 23.

The rest of the script iterates the same procedures, by grouping two Q-ID attributes of **(Y\_O\_D, RACE**). The rest of attributes will be anonymized by SSG\_P2, and stored in SG location. The final stage groups on (RACE), and anonymize the rest of the three attributes by NG\_P program. The remained records will be totally suppressed by SUP\_P. The previous procedures are repeated for each G group. For instance, in this example, the Seer data generates four G groups, as described in Table 4.3. The more created G groups, will lead to a better distribution of data in parallel. Table 4.3 is created and prepared before conducting any anonymization process. This is essential to building Pig script as per given information about Q-ID probabilities and the class values. The Pig script is an auto-generated script during the user’s access phase. This will be further discussed in the next chapter.

#### User-Defined Function

Six main UDF Java programs are defined in Pig Latin script: SSG\_P1, SSG\_P2, NG\_P, SUP\_P, OBV\_P, 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 few Q-ID bags do 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.

Java files are merged together in a Jar file implemented in UDF on anonymizing data. Six main Java files are created to execute the followings: masking, and ungrouping. In masking processes, there are three main algorithms implemented for taxonomy tree, interval, and suppression. The aim of any created UDF in MDSBA is keeping the algorithm humble with the minimal number of iterations and arrays. This is essential to reduce the data flow from the JVM reserved for YARN to the JVM reserved for UDF. Flowing a massive size of data to a black box of JVM may create Java Heap Memory error. Table 4.4 describes the Java programs needed for each G(QID) group.

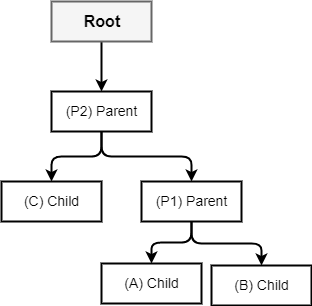
*Table 4.4- Illustrates the needed programs for each Q-ID group.*

|  |  |
| --- | --- |
| **Q-ID Group** | **Programs used** |
| G(QID)=2 | ADJUST.java, NG\_P.java, SUP\_P.java, OBV\_G |
| G(QID)=3 | ADJUST.java, **SSG\_P1.java**, NG\_P.java, SUP\_P.java, OBV\_G |
| G(QID)=4 | ADJUST.java, **SSG\_P1.java, SSG\_P2.java**, NG\_P.java, SUP\_P.java, OBV\_G |

Six Java programs are encapsulated in one JAR file. However, two main algorithms execute the primary masking operations in all anonymization tasks. Algorithms that mask taxonomy trees or intervals the major two algorithms among the six program.

#### Taxonomy Tree Anonymization Algorithm

Pig Latin deals with data bags, which are groups of tuples merged together by groupBy clause. In Pig Latin script, the JAR file is registered first at the top of the script. The registry command is given by (Register /jar\_path/file.jar). The taxonomy tree algorithm is implemented in three JAR programs; SSG\_P1, SSG\_P2, and NG\_P. The algorithm parse data from XML files available in the same JAR directory. The XML files include one XML file for taxonomy tree values, and other XML files for user’s parameters and information about data attributes. The taxonomy tree is structured from the top root element down to the parent’s elements. The XML is structured to simplify the reading by the anonymization algorithm. Two main nodes were created, root and branch. Each branch consists of children and parents. The children are located at the bottom of the node, while parents are presented by the rest of the tree nodes but the root. The taxonomy tree structure is shown in Figure 4.8.



*Figure 4.8- Taxonomy tree structure for XML files.*

Figure 4.8 illustrates the parents and children in each branch. Three branches can be abstracted from the illustrated taxonomy tree, these are branch1 ={A,P1, P2), branch2={B,P1,P2}, and branch3={C,P2}. However, in XML parsing, it is more reliable to create a similarity between XML nodes. Hence, all branches should contain an equivalent number of objects. In the previous example, each branch should contain 3 objects, so branch3 can be represented by three objects, branch3={C,P2,Root}. The object (Root) is added to create an equivalent number of objects in all branches. For better understanding, let us suppose a taxonomy tree for cars, as shown in Figure 4.9.

*Figure 4.9- Example of cars taxonomy tree.*

Figure 4.9 shows an example of car’s taxonomy tree with the probability for each node. Referring to Figure 4.8 and branches, an XML file can be generated as illustrated in Figure 4.10. The XML shows the root name and probability value at the top of the file, while branches nodes show both of children and parents. Children are given in a sequence of values in one sub-node, for <children names="KE series, AE series" />. The shown XML sample illustrates part of Figure 4.9 taxonomy. On anonymizing any value, the algorithm fetches all children nodes to find out the match word. If the anonymization algorithm decided to mask with parent 1, then parent 1 will be fetched depending on the child’s value. In the taxonomy example, suppose the following values were found in a dataset {AE series, MC series, Wagon, Lancer}. The anonymization was decided to be with parent 1, then the masking results will be: {Corolla, Camry, TE series, Mitsubishi}. The algorithm reads the number of parents from the node <parents\_no>3</parents\_no>, so the steps of anonymization can move up to 3 parents. For each parent the probability is compared with the sensitivity level ψ, if the probability is smaller or equal to ψ, then the parent is accepted, otherwise move to parent2. If parent2 was pointing to the (Root), then the root probability is compared again with ψ. If the sensitivity level is smaller than the probability value of the root, then the value is suppressed by using the word (Any).

The general algorithm initiates an array list by reading the data bag. The received data bag need to be anonymized by the taxonomy tree. The algorithm anonymizes all objects with parent 1, and checks the results. The results verification tends to count the number of equivalent objects after the first anonymization round. Any equivalent values are counted and verified if the count number is greater than or equal to . The rest of the non-equivalent values are further anonymized to the next parent. Once again the equivalent records are verified against the value of . The algorithm is shown in Figure 4.11.



*Figure 4.10- XML file example for the taxonomy tree in Figure 4.9.*

The algorithm verifies the number of objects and compares the number with. The algorithm is divided into two main sections; initial anonymization, and complete anonymization. The initial anonymization imposes a complete anonymization for each array object. The reading is pre-conditioned by comparison between each parent’s probability and the value of ψ. The initial anonymization is essential to accomplish most of the anonymization process. This may save a considerable time, if the data does not need any extra anonymization. The complete anonymization consists of two nested iterations. The first iteration allows a continuous loop until all objects match the equivalency of k anonymity. The loop process counts the similar objects, and tests if smaller than, if so, then another masking level is applied. The masking moves from parent 1 to parent 2 and so on, until reaching the root of the tree.

|  |
| --- |
| **Input**:array1={a1,a2,a3,…..an}, a∈ R: array objects  parents\_no=readXML(parents\_no) |
| **Processes**  //Anonymize all array objects to parent1 or higher (Initial Anonymization)  **Do While i < length\_of\_array**  //Check if parent probability ≤ ψ  FOR parent=1 TO parents\_no  parent\_prob=array1(i).readXML(parent)  parent\_value=array1(i).readXML(parent)  IF parent\_prob ≤ ψ  BREAK FOR  END IF  END FOR  //Update array1 with the anonymized values  array1(i)=replace\_anonyized\_object(parent\_value)  i=i+1  **END While**  //Arrange the array in ascending order  array1.ascend  Completed=false  All\_greater\_than\_k=true  i=1  count=1  //Keep anonymizing until k anonymity is applied on all array1 objects  **Do While completed = false**  //Another nested loop to scan array1 objects  **Do While i < length\_of\_array**  //IF statement to find the number of equivalent objects  **IF** array1(i)=array1(i-1)  count=count + 1  **ELSE IF** count <  All\_greater\_than\_k=false  **//anonymize to a higher parent**  **IF** parent ≤ parents\_no  parent\_value=array1(i-1).readXML(parent)  //Update the new anonymized value in a reverse order  h=i  **Do While h > i – count**  array1(h-1)=replace\_anonyized\_object(parent\_value)  h=h-1  **END While**  //Arrange the array in ascending order  array1.ascend  **ELSE**  parent\_value= array1(i-1).readXML(root\_name) // Anonymize by the root  //Update the new anonymized value in a reverse order  h=i  **Do While h > i – count**  array1(h-1)=replace\_anonyized\_object(parent\_value)  h=h-1  **END While**  //Arrange the array in ascending order  array1.ascend  **END IF**  **END IF**  //This condition is for verifying whether to stop the loop or continue  **IF** All\_greater\_than\_k=true **AND** i= length\_of\_array-1  Completed = true  **END IF**  All\_greater\_than\_k=true  i=i+1  count=1  **END While**  **END While** |
| **Output:** array1={A1, A2, A3 ….An} |

*Figure 4.11 Taxonomy masking algorithm.*

The algorithm can be summarized in the following:

* + - 1. Mask all array objects to parent 1
      2. Verify if parent 1 probability ≤ ψ
      3. If parent 1 probability > ψ then move to parent 2. Continue moving until reaching the root
      4. If the root probability > ψ, then suppress by (Any)
      5. After anonymizing the array, arrange the array in an ascending order
      6. Loop continuously until completed = true
      7. Scan the array objects, count the equivalency, if the equivalency is smaller than , then anonymize to the next parent
      8. Continue scanning until there is no equivalent records smaller than

The UDF program anonymizes the taxonomy tree values, and concatenates them with the rest of the attributes. The program inputs the data bag with compressed tuples and returns anonymized tuples to the Pig program. For a better understanding of the UDF anonymization procedures, let us consider a substance of Seer data as shown in Table 4.2. For this example, let us omit the Age attribute, while keeping the rest of the attributes: Year of Diagnosis, County, Race, and the class is Diagnostic. A data sample for this data set is shown below:

*Table 4.2- Sample of Seer data.*

|  |  |  |  |
| --- | --- | --- | --- |
| **YEAR OF DIAGNOSIS** | **COUNTY** | **RACE** | **Diagnostic** |
| 2010 | MI: Oakland (26125) | Black | Positive histology |
| 2010 | MI: Oakland (26125) | Black | no positive histology |
| 2010 | MI: Wayne (26163) | Black | Positive histology |
| 2013 | AL: Montgomery (36043) | White | Positive microscopic |
| 2013 | AL: Montgomery (36043) | White | no positive histology |
| 2013 | AL: Montgomery (36043) | White | Positive histology |
| 2013 | KY: Jefferson (21111) | White | no positive histology |

The dataset sample is grouped for anonymization with =3. The first input of data bag contains the first three records with the following format {2010, [MI: Oakland (26125), MI: Oakland (26125), MI: Wayne (26163)], Black, [Positive histology, no positive histology, Positive histology]}. The grouping was applied on both of: Race and Year of Diagnosis. The anonymization is applied on the lowest probability value, which is County. The UDF program masks the County to Parent1 of the taxonomy tree that is (Michigan). Luckily, the North state can be generalized to include the three tuples. However, the next four tuples cannot be generalized to Parent1. Since Parent1 for Montgomery is given by (Alabama), while Parent1 for Jefferson is given by (Kentucky). Also, moving up to Parent2 does not resolve the equivalency. Therefore, the masking moves up to the root of the tree, which is (USA State).

The previous example can be anonymized by following the algorithm steps. Firstly, all County objects are masked to Parent1. Secondly, the probability of Parent1 is compared with ψ. If the probability is ≤ ψ, then the first masking step outputs the following values; for the first group County(Parent1)={Michigan, Michigan, Michigan} and for the second group County(Parent1)={Alabama, Alabama, Alabama, Kentucky}. Next, the algorithm organizes each array in an ascending order. The algorithm, then, loops continuously until gaining the full equivalency. The first group will be scanned once to verify the equivalency. The algorithm used for this reason is:

IF array1(i)=array1(i-1)

count=count + 1

ELSE IF count <

As a result, the first group will not be further anonymized. The second group contains one dissimilar value (Kentucky), which should be similar to the rest of the counties. To target this value, the second group will be scanned and masked to a higher parent, County(Parent2)={South State, South State, South State, Southeast State}. As noticed, generalizing to a higher parent did not resolve the dissimilarity. Therefore, a higher generalization is required. The group is scanned and masked to a higher parent, because one object is dissimilar with the other three objects. Parent3 will be the root, and the final result of generalization is County(Root)={USA State, USA State, USA State, USA State}.

#### Interval Anonymization Algorithm

MDSBA implements User Defined Functions (UDF) in different locations. This is essential for two main purposes; anonymizing and ungrouping. In anonymizing, three masking types of interval, taxonomy tree, and suppression are implemented. Figure 4 shows the algorithm for anonymizing any numerical group. In the following algorithm, one numerical type attribute is used for anonymization. Minimizing the amount of data flowing to the UDF program is essential to reduce the processing cost and to avoid data overflow as described before.

|  |
| --- |
| **Input**: list={a1,a2,a3,…..an}, a∈ R: list is in ascending order |
| **Variables definitions**  var length\_of\_list=lists.size  var minimum=list(0)-list(0)%5  var range=1 / psi  var medium=minimum + range |
| **Processes**  //Loop to anonymize numerical  Do While object < length\_of\_list  object=object +1 //counter for objects in the list  //1. The main IF statement  IF list(object) >= minimum AND list(object) < medium THEN  rep=rep+1 //counter for objects within the interval  END IF  //2. Include the last few objects in the list within the same range  Count\_remained\_objects= length\_of\_list – object  IF Count\_remained\_objects < k\_dash THEN  medium=list(length\_of\_list-1)+(5-list(length\_of\_list-1)%5)    rep=rep + Count\_remained\_objects  object=object + Count\_remained\_objects  END IF  //3. Jump to the object that full fill k\_dash  IF rep < k\_dash THEN  Remain\_to\_k\_dash= k\_dash – rep  medium=list(object+ Remain\_to\_k\_dash-1)+(5-list(object+ Remain\_to\_k\_dash-1)%5)  IF medium – minimum < range THEN  medium =minimum + range  END IF  rep=rep + Remain\_to\_k\_dash+1  object=object + Remain\_to\_k\_dash  END IF  //4. Get the next object  IF list(object) > list(object-1) THEN  FOR I =0 TO rep    all\_intervals=all\_intervals+"["+minimum+" - "+medium+"[,"  NEXT I  minimum = medium  medium=medium + range  rep = 0 //reset the counter  END IF  END WHILE |
| **Output:** list={[A1-B1[,[A2-B2[,[A3-B3[,…,[An-Bn[} |
| *Figure 4.12 Algorithm illustrates the numerical values anonymization* |

It is hard to predict the behaviour of non-MapReduce JVM, but it is clear that we need to keep the data flow to the lowest level. For instance, JVM default installation may take up to 0.25 of the total memory. This size can be enlarged if needed. If MapReduce container memory is large enough to fit the data size, then the external JVM that handles the UDF may be able to handle a maximum of 25% of data size. On the contrast, MapReduce is able allocate a larger memory size, with a flexibility to spill any unfit data to the disk. Moreover, JVM created for UDF is not controlled by the cluster manager, therefore, processes are executed locally, and not within a distributed operations. The size of the UDF heap memory is not the only obstacle, but the complex iteration with several IF statements can be another cumbersome that degrades the data processes. MDSBA implements a swift algorithm to anonymize data with the minimal number of loops.

To understand the anonymization algorithm precisely, let us study the following example; a list of numerical values is given as: a list={2,3,4,6,12,12,12,18,25,26,26,30}, with =5. The algorithm anonymize this list as per ψ value. If ψ=0.2, then the range=1/0.2=5. The values are arranged in an ascending order. Referring to Figure 4.12, the program includes four sections. The program, first, initiates some values; length\_of\_list=10, minimum= 2 - (2 mod 5) =0, and medium =0+5 =5. In the first section of the program, the loop starts reading the first object (2), followed by the IF statement. Since the statement is true, then the counter is incremented by 1, rep=1. The next IF statement, in section 2, will be skipped, since the object list has not reached the end of it. In the third IF statement, the algorithm jumps by 4 objects, since Remain\_to\_k\_dash= k\_dash – rep=4. So the next list number will be 12, and the other parameters will be incremented for the next loop, rep=1+4+1=6 and object=5. In addition, the medium value is updated up to 15. Proceeding to the forth statement, it is clear that the 6th list number is also 12, so the statement result is false, and the loop iterates the second loop with object=6. In the second iteration, the first statement increments rep up to rep=7, while the statements two, three, and four are skipped. Similarly, the third loop increments rep by rep=8. In the third iteration, the number 18 exceeds the range, and the program skips to the forth statement. This is the first time that the forth statement is true, and the string of all\_intervals is generated by an iterated loop of 8 times, so the results will be new\_list{ [0-15[,[0-15[,[0-15[,[0-15[,[0-15[,[0-15[,[0-15[}. In this fourth statement the loop generates the new list, and both minimum and maximum are updated by minimum= 15, and maximum=15+5=20. The next loop updates the medium up to 35, so the final new list is updated by { [0-15[,[0-15[,[0-15[,[0-15[,[0-15[,[0-15[,[0-15[,[15-35[,[15-35[,[15-35[,[15-35[,[15-35[}.

## Comparison between Sensitivity-Based anonymization and Other Methods in Big Data

The first experiments are conducted on the Adult and Seer Datasets from the UCI Machine Learning Repository [25]. The data are public benchmarks for anonymization algorithms experiments. The experiments compare between MDSBA, Bottom-Up Generalization (BUG), and Multi-Dimensional Top-Down Specialization (MDTDS) in traditional and big data. Initially, the experiments tested the information gain and performance by small data size. Secondly, similar experiments are conducted by big data size. The aim of the experiments is establishing a comprehensive comparison between MDSBA and the other anonymization methods. Testing the information gain in any anonymized data is not a straightforward task. This can be applied by implementing public benchmarks such as; Naïve Gaussian and C4.5 classifiers.

Two separate scripts of Pig Latin were programmed. Two UDF Java programs were embedded in both of Pig script programs. The MDTDS and BUG algorithms rely intensively on Java UDF program, while the Pig script handles very limited tasks. The script aggregates the full-equivalent Q-ID attributes. The data bag with a number of records < k is transmitted to the UDF Java program. The algorithm calculates the scores for all Q-ID attributes. The highest attribute’s score will be specialized. For instance, if Education attribute gained the highest score, then the (Education) will be specialized from (Any) to (School, undergraduate, or postgraduate). Chapter two described the entropy, scores, and InfoGain equations. Also, Next section explains the implemented algorithm in Pig Latin script.

### UDF algorithm for MDTDS

The MDTDS method constrains UDF to operate intensively. This is because of the several iterations on calculating the best Q-ID score. Let us remember that the transmitted data was grouped by Pig Latin script. The Pig script groups by (Education, Sex, Age), and transmits data to UDF as a bag with a number of records < k. In UDF, the algorithm must group all Q-ID attributes but one. To understand the iteration process, let us study a test dataset from Adult data. The Adult dataset is given by three Q-IDs(Education, Age, Sex), as illustrated in Table 4.3. The UDF algorithm calculates the highest score among the three Q-IDs. The calculation imposes several times of aggregations. For instance, to calculate the Education score, the UDF algorithm receives the grouped data bag from Pig Latin script, ungroups the bag, and masks one of the Q-ID attributes by (ANY), as shown in Table 4.4-A. The algorithm counts the objects of Q-IDs and the class, then calculates the Education score. The score is calculated by the following steps:

* Mask Education by (ANY), as shown in Table 4.4-A
* Group by (Education, Sex, Age)
* Find the entropy of
* Mask Education by the first parent of the taxonomy tree. {School, University}, as in Table 4.4-B
* Find the entropy of
* Find the entropy of
* Find the InfoGain of
* Find the AnonyLoss of , where *A2* presents the number of records when EDU=Any, and *A1* presents the number of records when EDU=parent.
* Find the score of
* Repeat the previous steps for Sex, by masking Sex to (Any), and grouping by (Education, Sex, Age)
* Find the entropy of I(ANY), I(MALE), I(FEMALE). Finally find the score by calculating the InfoGain, AnonyLoss, and score.
* Repeat the previous steps for Age.
* Compare between score(EDU), score(SEX), and score(AGE).the highest score value will be specialized.
* Repeat the previous processes many times until there is no more cut left.

*Table 4.3- The original Adult data sample*

|  |  |  |  |
| --- | --- | --- | --- |
| **Education** | **Sex** | **Age** | **Salary (Class)** |
| 5th | Male | 20 | ≥50K |
| 10th | Male | 35 | <50K |
| 12th | Male | 36 | ≥50K |
| Bachelor | Female | 26 | <50K |
| Bachelor | Female | 33 | ≥50K |
| Master | Female | 42 | ≥50K |

*Table 4.4-A Adult data after generalizing EDU Table 4.4-B Adult data after EDU specializing*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Education** | **Sex** | **Age** | **Salary (Class)** |  | **Education** | **Sex** | **Age** | **Salary (Class)** |
| Any | Male | 20 | ≥50K | SCHOOL | Male | 20 | ≥50K |
| Any | Male | 35 | <50K | SCHOOL | Male | 35 | <50K |
| Any | Male | 36 | ≥50K | SCHOOL | Male | 36 | ≥50K |
| Any | Female | 26 | <50K | UNIVERSITY | Female | 26 | <50K |
| Any | Female | 33 | ≥50K | UNIVERSITY | Female | 33 | ≥50K |
| Any | Female | 42 | ≥50K | UNIVERSITY | Female | 42 | ≥50K |

As shown before, MDTDS implements an expensive computation process, by leaving the program with an unknown number of iterations. The program reads the data bag several times to reduce the AnonyLoss, by specializing the best Q-ID score. This algorithm may work fine with a small data sample. However, in big data, this technique is expensive and a time consumer. When there are millions of records, calculating the high scores, and finding the best cut is impractical. Instead, pre-defining the Q-ID for anonymization, and predetermining the best cut, before anonymization, can be a more reliable solution. MDTDS may perform better in traditional data, but its performance may degrade and becomes slow in big data.The anonymization of big data should focus on increasing the security and performance, rather than information gained. Neglecting the high accuracy of the best specializing cut will not dramatically affect the general statistical results. The statistical results are not affected by very miner values of anonymization loss.

### UDF algorithm for BUG

The applied algorithm in this UDF was derived from the advanced BUG (Adv-BUG) introduced in [26]. The procedures are close similar to the other BUG [27] and MRBUG [28] methods. Adv-BUG deliberately splits a random number of data groups, and then performs generalization as per Information Loss per Privacy Gain (ILPG). The implemented ILPG equations are similar to the previous MDTDS equations of entropy and InfoGain. The algorithm summary depends on multiple iterations of generalizations. The first stage of generalization applies Parent1 for each Q-ID attribute separately. For instance, in Adult data attributes (EDU, SEX, AGE), the algorithm applies generalization to EDU followed by InfoGain calculations. Next, the algorithm applies the generalization on SEX followed by InfoGain calculations. Finally, AGE is generalized and InfoGain is calculated and compared with the other two results. The accepted generalization among the three attributes is the one with the highest InfoGain result. In case if Parent1 generalization does not satisfy k-anonymity, the iteration moves generalization to the top of the taxonomy. The algorithm of the implemented BUG is described in Figure 4.13.

|  |
| --- |
| Algorithm BUG driver |
| **Input:** Data set D, anonymization level AL0, anonymity parameter k. |
| **Process:**  1. Scan data set D.  2. Initialize generalization by AL0 for each attribute  3. Calculate the values of search metric ILPG for each generalized attribute.  4. while ∃gen, gen < k  5. Identify the available generalization set AGSet out of all the active generalization candidates  6. Choose the generalization for the attribute with the highest InfoGain  7. If gen < k then  8. Generalize the QID attributes to Ali  9. Calculate the values of search metric ILPG for each generalized attribute  10. end while  11. Identify the available generalization set AGSet out of all the active generalization candidates  12. Choose the generalization for the attribute with the highest InfoGain |

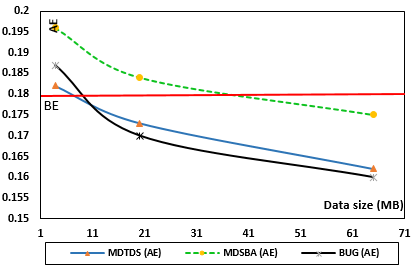
*Figure 4.13- The BUG driver algorithm implemented in UDF.*

### Small Data Size experiments

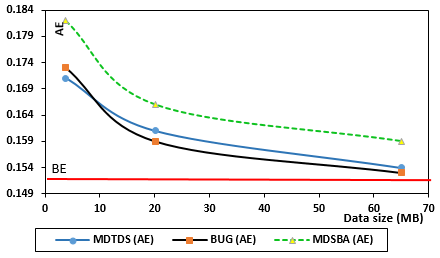
The first experiment aimed to compare between MDSBA BUG, and MDTDS in the traditional dataset. The used Adult and Seer datasets were divided into training and testing dataset. The chosen datasets were based on 70-30 ratio, where 70% of the source dataset for training model, and 30% of the source dataset for the testing model. Each dataset was enlarged to three different sizes: 3.8 MB, 20.1 MB, and 65 MB. The total number of records is 32,561 for the 3.8 MB, and 572,000 records for the 65 MB. 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 4.2 & 4.3. Adult data Q-IDs= {Age, Edu, Sex} and the class is salary= {<=50K, >50K}.

As mentioned earlier, the experiment was conducted by two Java programs embedded in two Pig Latin script. All experiments were conducted at the university Hadoop cluster, which contains of one NameNode and four DataNodes. Each node’s CPU is Intel(R) Xeon(R) CPU E5-2665 0 @ 2.40GHz x86\_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.

The first experiment implements a commonly available classifier for comparison between the three anonymization methods. The first comparison relies on Naïve Bayesian classifier. The anonymization was applied to two datasets, Adult and Seer datasets. After completing the anonymization processes for each dataset, the classification error was calculated by using RapidMiner Studio. The calculation was conducted before and after the anonymization. The experiment followed similar steps as in [29]. 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 4.13 and 4.14 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.



*Figure 4.13- Classification error for three sizes of Adult datasets*

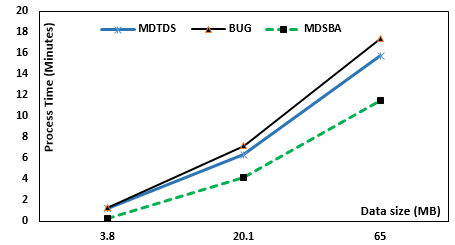


*Figure 4.14- Classification error for three sizes of Seer datasets*

In Figures 4.13 and 4.14, the classification error was much lower for MDTDS and BUG methods. The MDSBA method is not successful in the small size of datasets. This is related to the UDF algorithm for each method. The MDTDS iterates an unknown number of times to find the best Q-ID score. The best Q-ID score will be specialized, while the rest of the attributes are unspecialized. On each specializing process, scores calculation are conducted for each Q-ID attribute. This algorithm reduces the anonymization impact, but computationally expensive. BUG also, follows similar techniques. ILPG is calculated on each iteration time, and generalization is given according to InfoGain scores. On the other hand, MDSBA, pre-determines the anonymized Q-ID attribute, and pre-calculates the amount of anonymization applied. Hence, there is no need to iterate the program to make a decision, since the anonymized Q-ID is pre-determined, and the minimum generalization level is pre-calculated.

Figure 4.13 shows that AE drops below the BE line. Logically, this may indicate a low precision and recall of this classifier. The factual meaning of this diagram indicates that datasets may return better prediction after anonymization. This is practically possible since Naïve Bayesian and other classifiers measure the level of predictions but not the information gained or lost. This concern is a good reason to derive a better benchmark for measuring and comparing the anonymization loss in various anonymization methods. Nevertheless, the classifier accuracy, the aim of this experiment is comparing between MDSBA, BUG, and MDTDS. So far, the comparison shows more accuracy when anonymizing data by either MDTDS or BUG.

However, MDSBA computation cost is lower than BUG and MDTDS. Figure 4.15 illustrates the processing time for Seer cancer data. The diagram shows a comparison between the three methods performance as per minute. The results are expected as a reason for the low number of iteration and computation in MDSBA. It is apparent that reducing the computation cost may negatively affect the information gain percentage. MDSBA was not proposed for traditional data size. Instead it was developed to leverage the large data size, where the small numbers of accuracy may not change the final statistical results.



*Figure 4.15- Processing time comparison for the three anonymization methods*

### Large Data Size experiments

In the second experiment, datasets were enlarged to bigger data sizes. . Four different common data were used in this experiment 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 GB, 3.3 GB, 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 4.2 and 4.3. Adult data Q-IDs= {Age, Edu, Sex}, and the class contained the followings: salary= {<=50K, >50K}. Also, Heart Disease dataset is illustrated in Table 4.5, and Kasandr dataset is illustrated in Table 4.6. 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 five 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. The testing and training data were merged together during the anonymization process. The purchaseDate is generalized by a taxonomy tree of day, month, and year, as shown in Figure 4.16.

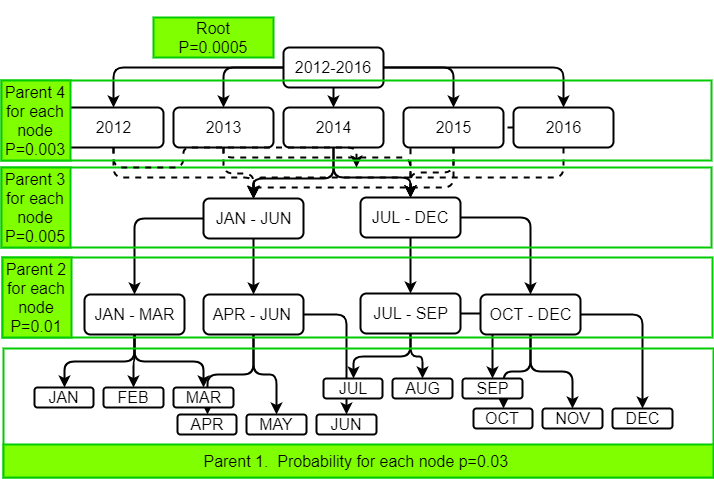
*Table 4.5- Heart Disease Q-ID attributes, propabilities and the class values.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Q-ID attributes** | | | | **CLASS** |
| ***Age P=0.01*** | ***Sex***  ***P= 0.33*** | ***Smoke***  ***P=0.5*** | ***CP (Chest Pain)***  ***P=0.25*** | ***class of electrocardiographic (restecg)*** |
| 1-100 | * Male * Female * Others | * Yes (1) * No (0) | * typical angina (0) * atypical angina (1) * non-angina pain (2) * asymptomatic (3) | * Normal (0) * Having ST-T wave abnormality (1) * Showing probable or definite left ventricular hypertrophy by Estes' criteria (2) |

*Table 4.6- Kasandr Q-ID attributes, propabilities, and the class values.*

|  |  |  |  |
| --- | --- | --- | --- |
| **Q-ID attributes** | | | **CLASS** |
| ***City***  ***P=0.0125*** | ***Category***  ***P=0.02*** | ***PurchaseDate***  ***P=0.0005*** | ***Merchant*** |
| * 80 cities in Germany | * 50 types | * 365 days in 5 years [taxonomy tree Figure 4.16] | * 738 types |

This experiment was implemented by Java as a UDF combined with Pig script. The script reads the database by using Hadoop reading process through HDFS. The experiment was divided into three sections; MDTDS, BUG, and MDSBA. Both methods, BUG and MDTDS, were executed several times to find out the best data size for Java Heap memory, since the large data size cannot be processed at once. To prevent Java Heap failure, the large file was split 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 4.7. Each split data is processed independently. 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 4.7. In MDSBA, the split is only needed when the k value is quite large, and the number of the class values is very small. As shown in Table 4.8, Adult data contains only two values of {<=50K, >50K}, so the data split has occurred when k ≥ 50. The rest of the data sets require less split operations. The increasing number of the class values may support logical split. The number of G groups increases parallel with the class values increase.



*Figure 4.16- Taxonomy tree for purchaseDate attribute in Kasandr dataset.*

*Table 4.7- Dataset of 4.6 GB split for both methods*

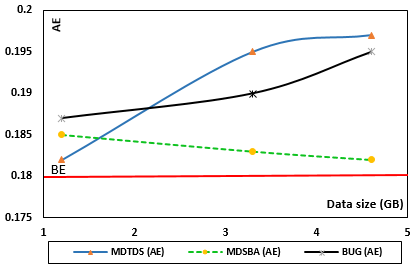
|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **K** | **4** | **8** | **12** | **16** | **20** | **50** | **100** | **170** | **250** | **300** | **400** |
| **MDTDS** | No Split | No Split | No Split | 1.15 GB | 650 MB | 420 MB | 230 MB | 164 MB | 110 MB | 73 MB | 38 MB |
| **BUG** | No Split | No Split | No Split | No Split | 1.15 GB | 770 MB | 570 MB | 350 MB | 230 MB | 170 MB | 95 MB |

*Table 4.8- Various Datasets of 4.6 GB split for MDSBA*

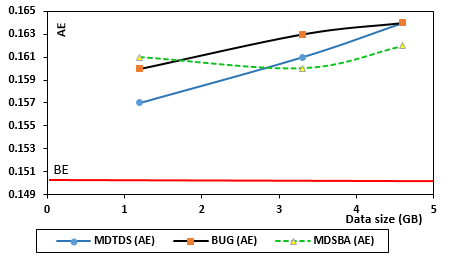
|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| K | 4 | 8 | 12 | 16 | 20 | 50 | 100 | 170 | 250 | 300 | 400 |
| **Adult** | No Split | No Split | No Split | No Split | No Split | 2.3 GB | 1.15 GB | 1.15 | 800 MB | 800 MB | 650 MB |
| **Seer** | No Split | No Split | No Split | No Split | No Split | No Split | No Split | 2.3 GB | 1.15 | 1.15 | 1.15 |
| **Heart Disease** | No Split | No Split | No Split | No Split | No Split | No Split | No Split | 2.3 GB | 1.15 | 1.15 | 1.15 |
| **Kasandr** | No Split | No Split | No Split | No Split | No Split | No Split | No Split | No Split | No Split | No Split | No Split |

The second experiment was implemented with a similar classifier to the first experiment. The aim is to compare the three anonymization methods. Moreover, more datasets have been added to a better evaluation. The calculation was conducted before and after the anonymization. Both BE and AE were measured for each dataset. The BE value are as follow: for Adult dataset 0.18, for Seer 0.15, for Heart Disease 0.16, and for Kasandr 0.17. Figures 4.17 to 4.20 show the classification error comparison between the three anonymization methods. The comparison considered k=50 in all trials. Three different data sizes were compared in the three anonymization methods.

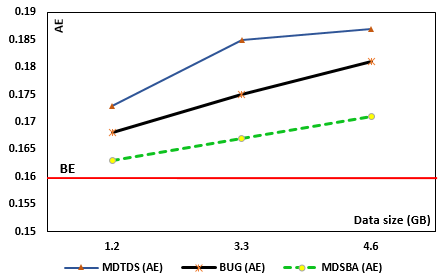
All datasets show similar results regarding AE values. The anonymization was conducted based on splitting data in BUG and MDTDS as shown in Table 4.7, and Table 4.8 for MDSBA split. All trials of this experiment showed a low AE in MDSBA, which is presented by a discrete line in the diagrams. Exceptionally, the two figures of 4.17 and 4.18 show a quite high AE value, when data size is 1.2 GB. This may indicate that BUG and MDTDS perform better than MDSBA, if no or few splits have occurred. In this trial the data size of 1.2 MB does not require many splits in the three anonymization methods, therefore, more accuracy and less AE values can be obtained when using BUG or MDTDS. However, MDSBA was proposed to increase the performance and security of big data. Fortunately, the information gained, after the anonymization with MDSBA, increases parallel with the data size increase. Figures 4.19 and 4.20 showed a lower AE value for MDSBA in comparison with the other methods. The low AE may indicate a high information gained in MDSBA. This may refer to the high number of data splits in BUG and MDTDS. Eventually, it was noticed that 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 cut is possible.



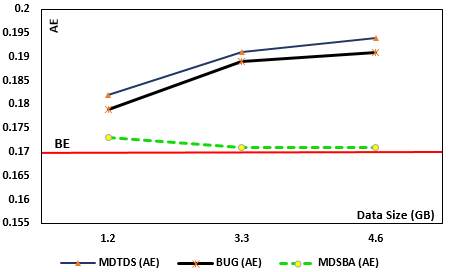
*Figure 4.17- AE in Adult dataset results for three different methods*



*Figure 4.18- AE results in Seer dataset for three different methods*



*Figure 4.19- AE results in Heart Disease dataset for three different methods*



*Figure 4.20- AE results in Kasandr dataset for three different methods*

## Anonymization Classification

Many techniques have been implemented to find a benchmark for measuring the performance of the anonymization methods. Techniques, usually compare data before and after anonymization to ascertain the information loss. The most widely used classifiers are Naïve Bayesian, C4.5, and K means [30]. The classifiers present algorithms to predict output results as per classified data. The comparison between pre-anonymization and post anonymization is measured by classification error by building a cost matrix and calculating the precision, and recall. This does not always mean an accurate measurement of data anonymization. Different classifiers may output different classification error. Hence, classifiers measure the prediction level rather than the actual anonymization level and information usefulness. In some previous experiments, the classification error of AE was lower than BE level, which is inadequate. For instance Figure 4.13 shows a drift of AE below the BE value. The BE value presents data before anonymization, while AE presents anonymized data. Various anonymization methods are expected to be proposed in the future, and a need for an adequate measurement tool is essential to compare them.

An alternative naïve equation is proposed to measure the percentages of information loss after anonymization, denoted by Disruption (Ɗ). The disruption value is a benchmark that gives a general indication of the size of anonymization loss. As shown in Equations 8 and 9, each anonymized block of tuples is calculated individually, and finally, the Ɗ value is the result of the total summation of all anonymized blocks. 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.

Ɗs (8)

Ɗ[total] (9)

Equation 8 is derived from the reverse proportion between the Q-ID probability and Ɗ. The Ɗ value increases with the increasing number of Q-ID attributes that participate in anonymization. Hence, anonymizing three Q-IDs will result in a higher Ɗ value than anonymizing two Q-IDs. Each block must have similar attribute values. If a block was found with two different anonymized values, then it will be split into more than one block. The following example illustrates Equations 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 Age. The number of anonymized records for these two blocks is 3 of each. The education anonymization was given parent 2, which relates to (certificate) and (degree), respectively. Both anonymized blocks are shown in Table 4.9. Based on the EDU taxonomy tree, as shown in the previous chapter; the first block probability is and the second block probability is. Also, the Age probability for each block is. Based on the previous two equations and the given information, the value of Ɗ is calculated as Ɗ1= (3×0.01) / (500×0.067×0.05) = 0.018, and Ɗ2= (3×0.01) / (500×0.17×0.05) =0.007. Referring to Equation 9, the total value of Ɗ = 0.025.

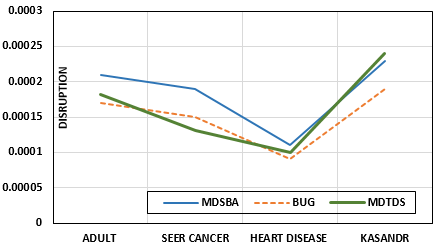
*Table 4.9- Disruption example with Adult data*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Blocks** | **Education** | **Sex** | **Age** | **Salary (Class)** |
| Block 1 | CERTIFICATE | Male | 20 - 40 | ≥50K |
| Block 1 | CERTIFICATE | Male | 20 - 40 | <50K |
| Block 1 | CERTIFICATE | Male | 20 - 40 | ≥50K |
| Block 2 | DEGREE | Female | 30 – 50 | <50K |
| Block 2 | DEGREE | Female | 30 – 50 | ≥50K |
| Block 2 | DEGREE | Female | 30 – 50 | ≥50K |

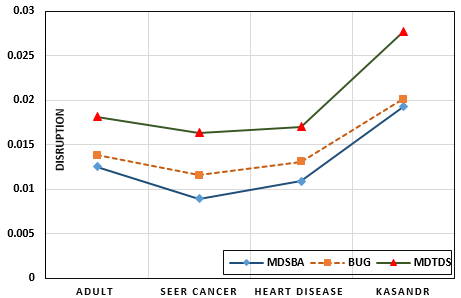
The above example shows a suggested equation for comparing between several anonymization methods. The equation can accurately measure the amount of obfuscation in any dataset. Each dataset may output different values of Ɗ, which merely depends on the data nature. Therefore, it is not possible to adopt one value of Ɗ as a benchmark for all data. However, lower Ɗ value may indicate a low obfuscation in data.

### Using Disruption to Compare between MDSBA and the Other Anonymization Methods

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. The second experiment recalled the 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 BUG, MDTDS, and MDSBA. 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 and 50. The value of k is used for BUG and MDTDS, while is used for MDSBA. Figure 4.21 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 may not have a real impact on data analytics. Figure 4.22, 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.



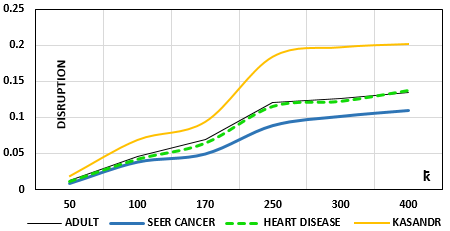
*Figure 4.21- Disruption comparison for =4 and the 4 datasets.*



*Figure 4.21- Disruption comparison for =50 and the 4 datasets.*

These results are expected, since the smaller value of k=4 does not require any split in BUG or MDTDS, as shown in Table 4.7. It was examined earlier that BUG and MDTDS gain better results in information usefulness, if no split has occurred. Therefore, Figure 4.21 shows a higher disruption value for MDSBA, while Figure 4.22 shows the lowest disruption value for MDSBA. For 4.6 GB data size, both of BUG and MDTDS need be split several times, which degrades the information usefulness.

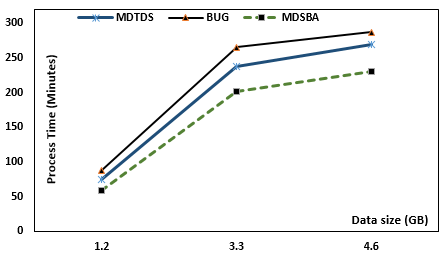
In the second experiment, the aim was finding the anonymization level in MDSBA on the variation of. Figure 4.23 shows a gradual increase in Ɗ on ≥ 50 for all dataset. This Ɗ increase is expected as a reason of the equivalency decrease with the large number of. This experiment was conducted with a data size of 4.6 GB for each dataset. The 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 the previous and current experiments, anonymization was applied on verities of datasets, and it was found that anonymizing most datasets will output a Ɗ below 5%. However, this finding is a rough estimate and cannot be generalized to all datasets. The value Ɗ is related to other factors, such as the data cumulative distribution, and the attributes probabilities, as described before.



*Figure 4.22- The relationship between Ɗ and variation in MDSBA.*

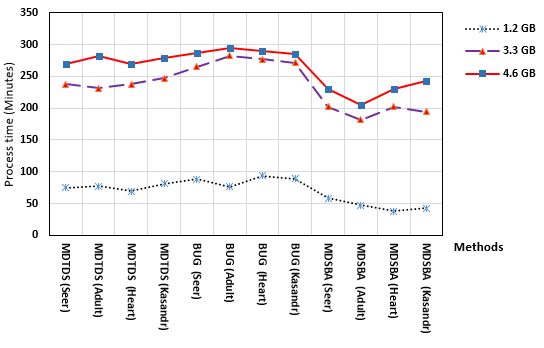
## Performance Comparison

Figure 4.15 illustrated a considerable time difference in processing the three anonymization methods. It showed the lowest processing time for MDSBA, followed by MDTDS, and finally BUG. The more accurate data after anonymization will require a more expensive process. The previous performance measurement was implemented in small data sizes of Seer data. In this experiment, let us apply larger size of data. The large size means quite large enough to be accommodated in the university Hadoop lap. The lap resources are limited, hence adding more data is not possible. In this experiment, a large data of Gigas was prepared and tested for processing time. The datasets used in this experiment are: 1.2 GB, 3.3GB, and 4.6 GB. The first part of the experiment has measured the Seer Cancer data performance. Similar to the small data size, the big data size also showed similar results, as illustrated in Figure 4.23.



*Figure 4.23- Processing time comparison for the three-anonymization methods*

The second part of the experiment showed a good performance for MDSBA. Figure 4.24 illustrates the decline of the three data sizes on anonymizing data with MDSBA. The four datasets showed quite similar results. This experiment was conducted on k=50. The value of k imposes several splits of data sizes for MDTDS and BUG. This split negatively affects the gained information, but it does not affect the processing time. BUG and MDTDS are computationally expensive, since their algorithm relies on iterating a large size of data block. As described earlier, the large size of data block is transmitted to the UDF program, which may be degraded by Java Heap memory limitations. Reducing the data flow to the UDF program is essential to leverage the parallelization process.



*Figure 4.24- Processing time comparison of various datasets and sizes*

## Summary

In this chapter, several experiments are conducted to measure and compare the MDSBA method. The experiments showed promising results in anonymizing big data by MDSBA method. The algorithms used in experiments have supported the state of the art for k-anonymity concept. The aim was comparing MDSBA with the other known anonymization methods, BUG and MDTDS. It was stated that both methods are proper for small data sizes, while they are not applicable in large data sizes without several splits. However, it was proven that the several splits have a negative impact on gaining information usefulness. Also, it was proven that data with small k-anonymity value impose less number of splits. This is because of the high equivalency percentage in small values of k. Generally, MDSBA is best fit the larger datasets with a quite large value of k.

It was experimentally proven the good performance of MDSBA in comparison with BUG and MDTDS. The MDSBA processing time showed a low cost in small and large data size. This is because of the pre-calculated and pre-determined anonymization parameters. In MDSBA, the anonymized Q-ID and the amount of masking are computed prior the anonymization process. Moreover, the three anonymization methods (BUG, MDSBA, and MDTDS) are implemented by Pig Latin scripts and UDF. The UDF algorithms are created and tested for comparison purposes. It was explained that BUG and MDSBA intensively rely on the UDF program instead of the parallelization process. Also, the UDF, in both methods, receive a massive data flow during the anonymization process. This may create an extra cumbersome on executing the UDF algorithm. UDF receives a large size of data blocks, and iterates, these large blocks, several times.

The classification matrices were implemented to compare the prediction before and after the anonymization. It is clear that the classifiers focus on the prediction, precision, recall and classification error, instead of focusing on the information gain before and after anonymization. For a better benchmark for any future comparison between proposed methods, this chapter introduced a disruption D equation. The equation can measure the exact amount of obfuscation on data after anonymization

[1] S. Chaudhuri, "What next?: a half-dozen data management research goals for big data and the cloud," in *PODS '12*, M. Lenzerini, M. Benedikt, Kr, and M. tzsch, Eds., ed: ACM, 2012, pp. 1-4.

[2] Q. Tran and H. Sato, "A solution for privacy protection in mapreduce," ed, 2012, pp. 515-520.

[3] X. Z. Xindong Wu, Gong-Qing Wu,Wei Ding, "Data Mining with Big Data," *IEEE,* 2014.

[4] L. T. Y. Xuyun Zhang, Chang Liu, Jinjun Chen,, "A Scalable Two-Phase Top-Down Specialization Approach for Data Anonymization Using MapReduce on Cloud," *IEEE,* 2014.

[5] Y. Sowmya and M. Nagaratna, "Parallelizing K-Anonymity Algorithm for Privacy Preserving Knowledge Discovery from Big Data," *International Journal of Applied Engineering Research,* vol. 11, pp. 1314-1321, 2016.

[6] T. White, *Hadoop The Definitive Guide*, 4th ed. ed. Sebastopol: Sebastopol : O'Reilly Media, 2015.

[7] B. Lakhe, "Open Source Authentication in Hadoop," in *Practical Hadoop Security*, ed: Springer, 2014, pp. 51-74.

[8] Dhruba Borthakur, "The Hadoop Distributed File System:

Architecture and Design," *apache,hadoop.,* 2014.

[9] D. Park, K. Kang, J. Hong, and Y. Cho, "An efficient Hadoop data replication method design for heterogeneous clusters," in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, 2016, pp. 2182-2184.

[10] K. Li, C. Reichenbach, Y. Smaragdakis, Y. Diao, and C. Csallner, "SEDGE: Symbolic example data generation for dataflow programs," in *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, 2013, pp. 235-245.

[11] A. Bali, "A Review of Hadoop Ecosystems," 2016.

[12] P. Zikopoulos and C. Eaton, *Understanding big data: Analytics for enterprise class hadoop and streaming data*: McGraw-Hill Osborne Media, 2011.

[13] A. Rheinländer, U. Leser, and G. Graefe, "Optimization of Complex Dataflows with User-Defined Functions," *ACM Computing Surveys (CSUR),* vol. 50, p. 38, 2017.

[14] S. Bhargav, *Troubleshooting Ubuntu Server*: Packt Publishing Ltd, 2015.

[15] P. Manuel, "A trust model of cloud computing based on Quality of Service," *Annals of Operations Research,* vol. 233, pp. 281-292, 2015.

[16] O. O’Malley, K. Zhang, S. Radia, R. Marti, and C. Harrell, "Hadoop security design," *Yahoo, Inc., Tech. Rep,* 2009.

[17] B. Spivey and J. Echeverria, *Hadoop Security: Protecting your big data platform*: " O'Reilly Media, Inc.", 2015.

[18] P. P. Sharma and C. P. Navdeti, "Securing big data hadoop: a review of security issues, threats and solution," *Int. J. Comput. Sci. Inf. Technol,* vol. 5, pp. 2126-2131, 2014.

[19] S. Narayanan, *Securing Hadoop*: Olton : Packt Publishing, 2013., 2013.

[20] P. Pasupuleti, *Pig Design Patterns*. Birmingham: Birmingham : Packt Publishing, 2014.

[21] A. Schätzle, M. Przyjaciel-Zablocki, and G. Lausen, "PigSPARQL: Mapping SPARQL to pig latin," in *Proceedings of the International Workshop on Semantic Web Information Management*, 2011, p. 4.

[22] M. Lutz, *Programming python* vol. 8: O'Reilly, 1996.

[23] M. Grimmer, "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*, 2014, pp. 17-19.

[24] D. Koenig, A. Glover, P. King, G. Laforge, and J. Skeet, *Groovy in action* vol. 1: Manning, 2007.

[25] R. K. a. B. Becker. (1996). *Adults Data*. Available: <ftp://ftp.ics.uci.edu/pub/machine-learning-databases>

[26] K. Pandilakshmi and G. R. Banu, "An Advanced Bottom up Generalization Approach for Big Data on Cloud," *Volume,* vol. 3, pp. 1054-1059, 2014.

[27] P. S. Ke Wang, S. Yu, and S. Chakraborty, "Bottom-up generalization: a data mining solution to privacy protection," ed. USA, 2004, pp. 249-256.

[28] A. Irudayasamy and L. Arockiam, "Parallel bottom-up generalization approach for data anonymization using map reduce for security of data in public cloud," *Indian Journal of Science and Technology,* vol. 8, p. 1, 2015.

[29] B. C. M. Fung, K. Wang, and P. S. Yu, "Top-down specialization for information and privacy preservation," ed. USA, 2005, pp. 205-216.

[30] M. Guller, *Big Data Analytics with Spark A Practitioner's Guide to Using Spark for Large Scale Data Analysis*: Berkeley, CA : Apress : Imprint: Apress, 2015., 2015.