

**Protecting Privacy in Big Data by Implementing Sensitivity-based Anonymity Framework**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Mohammed Al-Zobbi

A thesis submitted in fulfilment of the degree of Doctor of Philosophy

School of Computing, Engineering and Mathematics

Western Sydney University

March 2018

# ABSTRACT

Big data is predominantly associated with data retrieval, storage, and analytics. The world is creating a massive data size, which increases exponentially. Since dawn of time until 2015, human had created 7.9 Zettabyte. This number will be exponentially raised up 40.9 Zettabyte by 2020. Analytics in big data is maturing and moving towards mass adoption. The emergence of analytics increases the need for innovative tools and methodologies to protect data against privacy violation. Data analytics is prone to privacy violations and data disclosures, which can be partly attributed to the multi-user characteristics of big data environments. Adversaries may link data to external resources, try to access confidential data, or deduce private information from the large number of data pieces that they can obtain. Many data anonymisation methods were proposed to provide some degree of privacy protection by applying data suppression and other distortion techniques. However, currently available methods suffer from poor scalability, granularity, performance, and lack of framework standardization. Current anonymisation methods are unable to cope with the massive size of data processing. Some of these methods were especially proposed for MapReduce framework to operate in big data. However, they still operate in conventional data management approaches. Therefore, there were no remarkable gains in the performance.

To fill this gap, this thesis introduces a framework that can operate in MapReduce environment to benefit from its advantages, as well as from those in Hadoop ecosystems. The framework provides a granular user’s access that can be tuned to different authorization levels. The proposed solution provides a fine-grained alteration based on the user’s authorization level to access MapReduce domain for analytics. The framework core concept was derived from data k-anonymisation, which was proposed by Sweeney on 1998. Using well-developed role-based access control approaches, this framework is capable of assigning roles to users and mapping them to relevant data attributes. Moreover, the thesis introduces a simple classification technique that can accurately measure the anonymisation extent in any anonymized data. Various expirments show promising results on applying the framework porposed in this thesis. The framework anonymisation expirements output graduality, parallel with a high scalability and a low distortion.

To confirm the effectiveness of the proposed framework in protecting privacy and reducing data loss, a diverse range of experimental studies are carried out. The experimental studies aimed to demonstrate the capability of the framework granularity by applying gradual levels of anonymisation for data analysers. The experiments meant to compare between the proposed anonymisation framework and the current available frameworks. The comparisons are related to performance and data loss in big data operation tools, such as MapReduce and Spark. The experiment’s results showed higher performance output, when anonymisation was conducted in Spark. However, in some limited cases, MapReduce is preferable when the cluster resources are limited and the network is non-congested.

The experiments unveil several facts regarding big data behaviour. For instance, big data tends to be more equivalent as the data size increase. Moreover, the major big data concern is security, hence, focusing on security side should be the main target. The little obfuscated records do not have a major impact on the all over statistical results. Therefore, the trade-off between security and information gain tends to give security a higher priority. It is expected that big data access is requested by many users. This massive demand has recently increased with the social media blossom over the Internet. Personal and contextual information are available online publicly. Thus, personal re-identification has never been easier than now. For this reason, security should be the major focus of anonymisation algorithms.

The experiments also show a high performance and an average information loss for the proposed anonymisation framework. The anonymised data has gained a low classification error by Bayesian classifier. In comparison to the current anonymisation methods, the proposed framework has a little lower classification error by 0.12%. In performance wise, the proposed framework has reached up to 40% faster than the current anonymisation frameworks. The security side was strengthened by increasing the *k-anonymity* value, and assigning granularity for user’s access.

# Table of Contents

[ABSTRACT I](#_Toc507306462)

[Table of Contents III](#_Toc507306463)

[STATEMENT OF AUTHENTICATION VIII](#_Toc507306464)

[ACKNOWLEDGEMENTS IX](#_Toc507306465)

[LIST OF FIGURES X](#_Toc507306466)

[LIST OF TABLES XII](#_Toc507306467)

[LIST OF ACRONYMS XIV](#_Toc507306468)

[1 - INTRODUCTION 1](#_Toc507306469)

[1.1 Research Question 4](#_Toc507306470)

[1.2 Thesis objective 5](#_Toc507306471)

[1.3 Thesis Contribution 5](#_Toc507306472)

[1.4 Thesis Layout 7](#_Toc507306473)

[2 - BACKGROUND AND CHALLENGES 11](#_Toc507306474)

[2.1 Big Data Structure and Challenges 12](#_Toc507306475)

[2.1.1 Data Analytics 16](#_Toc507306476)

[2.2 Big Data analytics Challenges 18](#_Toc507306477)

[2.2.1 MapReduce and Hadoop 20](#_Toc507306478)

[2.2.2 Hadoop Ecosystems 23](#_Toc507306479)

[2.2.3 Streaming and Spark 24](#_Toc507306480)

[2.3 Security Challenges in Big Data Analytics 27](#_Toc507306481)

[2.3.1 Protecting Privacy by Differential Privacy 28](#_Toc507306482)

[2.3.1.1 Differential Privacy Frameworks 30](#_Toc507306483)

[2.3.1.1.1 Airavat 30](#_Toc507306484)

[2.3.1.1.2 GUPT 31](#_Toc507306485)

[2.3.1.2 Possible attacks in Differential Privacy 32](#_Toc507306486)

[2.3.2 Protecting Privacy by *K-anonymity* 33](#_Toc507306487)

[2.3.2.1 Impairments in *K-anonymity* 33](#_Toc507306488)

[2.4 *K-anonymity* Frameworks 37](#_Toc507306489)

[2.4.1 *K-anonymity* Methods for Traditional Data 37](#_Toc507306490)

[2.4.1.1 Generalization 37](#_Toc507306491)

[2.4.1.1.1 Incognito 40](#_Toc507306492)

[2.4.1.1.2 *ℓ-diversity* 41](#_Toc507306493)

[2.4.1.2 Specialization 43](#_Toc507306494)

[2.4.1.2.1 LKC Privacy 45](#_Toc507306495)

[2.4.1.2.2 (*α, k*)-anonymisation 47](#_Toc507306496)

[2.4.2 Critic of Traditional Data Anonymisation Methods 49](#_Toc507306497)

[2.4.3 *K-anonymity* Methods for Big Data 50](#_Toc507306498)

[2.4.3.1 Generalization 50](#_Toc507306499)

[2.4.3.1.1 Advanced BUG 50](#_Toc507306500)

[2.4.3.2 Specialization 51](#_Toc507306501)

[2.4.3.2.1 Two-Phase Top-Down Specialization 51](#_Toc507306502)

[2.4.4 Critic of Big Data Anonymisation Methods 52](#_Toc507306503)

[2.5 Summary 54](#_Toc507306504)

[3 - MULTI-DIMENTIONAL SENSITIVITY-BASED ANONYMISATION METHOD 56](#_Toc507306505)

[3.1 Requirements for Big Data Anonymisation Method 57](#_Toc507306506)

[3.1.1 Equivalency Increase 57](#_Toc507306507)

[3.1.2 Information Gain and Security 60](#_Toc507306508)

[3.1.3 Parallel Algorithm 62](#_Toc507306509)

[3.1.4 Gradual Access 63](#_Toc507306510)

[3.2 Multi-Dimensional Sensitivity-Based Anonymisation Method Concept 63](#_Toc507306511)

[3.2.1 Quasi Identifiers and Classes 64](#_Toc507306512)

[3.2.2 Probability Concept and Anonymisation Masking 64](#_Toc507306513)

[3.2.2.1 Interval and Taxonomy Tree Masking 66](#_Toc507306514)

[3.2.2.2 Suppression Masking 68](#_Toc507306515)

[3.2.3 Grouping Data Vertically and Horizontally 70](#_Toc507306516)

[3.2.3.1 Grouping Data Vertically 70](#_Toc507306517)

[3.2.3.2 Grouping Data Horizontally 72](#_Toc507306518)

[3.2.4 Mathematical Equations to Calculate the Sensitivity Level 74](#_Toc507306519)

[3.2.4.1 Sensitivity Level and Time Factor 76](#_Toc507306520)

[3.2.4.2 Anonymisation Operations 78](#_Toc507306521)

[3.2.4.3 Taxonomy Tree Masking 83](#_Toc507306522)

[3.2.4.4 Discretization Masking 85](#_Toc507306523)

[3.3 Summary 86](#_Toc507306524)

[4 - implementing Sensitivity-Based Anonymisation by Hadoop Ecosystems 87](#_Toc507306525)

[4.1 Hadoop in Data Analytics 88](#_Toc507306526)

[4.1.1 Hadoop Core 89](#_Toc507306527)

[4.4.1. Hadoop Ecosystems 90](#_Toc507306528)

[4.2 Hadoop security 91](#_Toc507306529)

[4.2.1 Implementing Hadoop in LDAP Domain 92](#_Toc507306530)

[4.2.2 Applying Kerberos for Hadoop Secure Mode 93](#_Toc507306531)

[4.3 Deploying Sensitivity-Based Anonymisation by using Hadoop Ecosystems 95](#_Toc507306532)

[4.3.1 Pig Core Structure 95](#_Toc507306533)

[4.3.2 Anonymisation by Pig Latin scripts 99](#_Toc507306534)

[4.3.2.1 Pig Latin Script Example 103](#_Toc507306535)

[4.3.2.2 User-Defined Function 106](#_Toc507306536)

[4.3.2.3 Taxonomy Tree Anonymisation Algorithm 107](#_Toc507306537)

[4.3.2.4 Interval Anonymisation Algorithm 114](#_Toc507306538)

[4.4 Comparison between Sensitivity-Based anonymisation and Other Methods in Big Data 116](#_Toc507306539)

[4.4.1. UDF algorithm for MDTDS 116](#_Toc507306540)

[4.4.1 UDF algorithm for BUG 118](#_Toc507306541)

[4.4.2 Small Data Size experiments 119](#_Toc507306542)

[4.4.3 Large Data Size experiments 122](#_Toc507306543)

[4.5 Anonymisation Classification 127](#_Toc507306544)

[4.5.1 Using Disruption to Compare between MDSBA and the Other Anonymisation Methods 128](#_Toc507306545)

[4.6 Performance Comparison 130](#_Toc507306546)

[4.7 Summary 132](#_Toc507306547)

[5 - Framework for Sensitivity-Based Anonymisation 134](#_Toc507306548)

[5.1 Security Assertion Markup Language (SAML) 135](#_Toc507306549)

[5.2 MDSBA and Granular Access Control 138](#_Toc507306550)

[5.2.1 Live Data and Archived Data 139](#_Toc507306551)

[5.2.2 MDSBA *k̄* percentage and Business roles 142](#_Toc507306552)

[5.2.3 MDSBA Three Services 144](#_Toc507306553)

[5.2.3.1 Core service 147](#_Toc507306554)

[5.2.3.1.1 Keep or Remove Decision 149](#_Toc507306555)

[5.2.3.1.2 Initial Access 150](#_Toc507306556)

[5.2.3.1.3 Subordinate Access 151](#_Toc507306557)

[5.2.3.2 Initialiser Service 151](#_Toc507306558)

[5.2.3.2.1 Generating Pig scripts in Initialiser Service 154](#_Toc507306559)

[5.2.3.3 Anonymiser Service 156](#_Toc507306560)

[5.2.3.3.1 Service Provider Log File and Purge Process 159](#_Toc507306561)

[5.3 Improvement to MDSBA Security 162](#_Toc507306562)

[5.3.1. Obvious Guess 162](#_Toc507306563)

[5.3.2. Across Groups Unique Identifiers (AGUI) 163](#_Toc507306564)

[5.3.3. Resolving Obvious Guess 164](#_Toc507306565)

[5.3.4. Resolving Across Groups Unique Idntifier (AGUI) 165](#_Toc507306566)

[5.4 Experimenting Data Disruption in MDSBA Framework 166](#_Toc507306567)

[5.5 Summary 170](#_Toc507306568)

[6 - Towards Optimal *k-anonymity* 172](#_Toc507306569)

[6.1 Previous Solution to Find the Optimal *k* Value 173](#_Toc507306570)

[6.2 MDSBA Grouping and the Gradual Access 174](#_Toc507306571)

[6.3 Possible Attacks against MDSBA 176](#_Toc507306572)

[6.4 Finding the optimal *k* value 177](#_Toc507306573)

[6.4.1 Cumulative Frequency 178](#_Toc507306574)

[6.4.2 Linear Regression 180](#_Toc507306575)

[6.4.2.1 Three security levels 183](#_Toc507306576)

[6.5 Finding the Optimal *k* percentage 184](#_Toc507306577)

[6.6 Dynamic G(QID) groups 188](#_Toc507306578)

[6.7 Summary 192](#_Toc507306579)

[7 - Comparison between MapReduce and Spark 194](#_Toc507306580)

[7.1 Analytics in Big Data 195](#_Toc507306581)

[7.2 Spark Structure 197](#_Toc507306582)

[7.3 MapReduce and Spark 202](#_Toc507306583)

[7.4 Data Streaming versus Batch 206](#_Toc507306584)

[7.5 Implementing MDSBA in Spark 207](#_Toc507306585)

[7.5.1 User Defined Function in MDSBA 210](#_Toc507306586)

[7.5.2 Differences between Pig and Spark Algorithms 212](#_Toc507306587)

[7.6 Comparison between Hadoop ecosystems and Spark 217](#_Toc507306588)

[7.6.1 Spark Tuning in MDSBA 221](#_Toc507306589)

[7.7 Summary 224](#_Toc507306590)

[8 - CONCLUSION 225](#_Toc507306591)

[References 230](#_Toc507306592)

# STATEMENT OF AUTHENTICATION

I declare that to the best of my knowledge the work described in this thesis is, except where otherwise stated, entirely my own work and has not been submitted for a degree at this or any other university.

X

# ACKNOWLEDGEMENTS

I would like to thank my supervisor Dr. Seyed Shahrestani for his academic guidance, motivation, suggestions, and support throughout my entire thesis and for his administrative and logistic support during my study, including my travelling, and providing me with supportive tools and devices. Without his continuous help and support, it would not have possible for me to complete this thesis. I am deeply indebted to him for all his help and support.

I also thank my co-supervisor Dr. Chun Ruan for her continuous help in completing this thesis. She was deeply involved in the theoretical and practical experiments and theories, which motivated my skills in brainstorming and literal thinking. She has always checked my work and provided me fruitful feedback to improve my work. I appreciate all her help and support.

I would like to express my cordial gratitude to all my colleagues in the Networking, Security, and Technical Support Department for their support throughout my study. I also appreciate the help and support of other colleagues from the School of Computing, Engineering and Mathematics.

A special thanks goes to my wife Ling for standing beside me throughout my study and research time. She has been my inspiration and motivation for continuing to improve my knowledge and move my study forward. I also thank my two wonderful children: Salahudeen, and Malek, for always making me smile.

# LIST OF FIGURES

[Figure ‎2.1- The three layers of big data structure 14](#_Toc508099265)

[Figure ‎2.2- Data map shows the patient’s information distribution -source [43] 18](#_Toc508099266)

[Figure ‎2.3-YARN structure in MapReduce 21](#_Toc508099267)

[Figure ‎2.4- YARN supports an effective share of resources. 22](#_Toc508099268)

[Figure ‎2.5- High-Level architecture of Airavat 29](#_Toc508099269)

[Figure ‎2.6- Taxonomy Trees for JOB, AGE, and SEX 45](#_Toc508099270)

[Figure ‎3.1- Proportionality of equivalent records with increased record numbers 59](#_Toc508099271)

[Figure ‎3.2- The effect of increasing the number of actual combinations with increased number of records 59](#_Toc508099272)

[Figure ‎3.3- Taxonomy tree for EDU in Adult data. 66](#_Toc508099273)

[Figure ‎3.4- Access Control structure in MDSBA 71](#_Toc508099274)

[Figure ‎3.5- Plotted graph to derive Equation 5 for aging factor 77](#_Toc508099275)

[Figure ‎3.6- MDSBA algorithm 80](#_Toc508099276)

[Figure ‎4.1- Comparison between Hadoop v1 and v2 89](#_Toc508099277)

[Figure ‎4.2- Kerberos and LDAP server in MDSBA 93](#_Toc508099278)

[Figure ‎4.3- Kerberos general steps to secure services by Administrators 95](#_Toc508099279)

[Figure ‎4.4. Illustrates Pig Latin structure 98](#_Toc508099280)

[Figure ‎4.5-Pig structure at the top of the processing framework 99](#_Toc508099281)

[Figure ‎4.6- Four Processing stages to anonymise four Q-IDs group 102](#_Toc508099282)

[Figure ‎4.7-A. Anonymisation process of three Q-IDs. Figure 4.7-B. Anonymisation process for two Q-IDs 103](#_Toc508099283)

[Figure ‎4.8- Pig Latin script example for the first two stages 105](#_Toc508099284)

[Figure ‎4.9- Taxonomy tree structure for XML files 109](#_Toc508099285)

[Figure ‎4.10- Example of cars taxonomy tree. 110](#_Toc508099286)

[Figure ‎4.11- XML file example for the taxonomy tree in Figure 4.10 111](#_Toc508099287)

[Figure ‎4.12- Taxonomy masking algorithm 113](#_Toc508099288)

[Figure ‎4.13- Algorithm illustrates the numerical values anonymisation 116](#_Toc508099289)

[Figure ‎4.14- The BUG driver algorithm implemented in UDF 121](#_Toc508099290)

[Figure ‎4.15- Classification error for three sizes of Adult datasets 123](#_Toc508099291)

[Figure ‎4.16- Classification error for three sizes of Seer datasets 123](#_Toc508099292)

[Figure ‎4.17- Processing time comparison for the three anonymisation methods 124](#_Toc508099293)

[Figure ‎4.18- Taxonomy tree for purchase Date attribute in Kasandr dataset. 126](#_Toc508099294)

[Figure ‎4.19- AE in Adult dataset results for three different methods 128](#_Toc508099295)

[Figure ‎4.20- AE results in Seer dataset for three different methods 128](#_Toc508099296)

[Figure ‎4.21- AE results in Heart Disease dataset for three different methods 129](#_Toc508099297)

[Figure ‎4.22- AE results in Kasandr dataset for three different methods 129](#_Toc508099298)

[Figure ‎4.23- Disruption comparison for =4 and the 4 datasets 132](#_Toc508099299)

[Figure ‎4.24- Disruption comparison for =50 and the 4 datasets 132](#_Toc508099300)

[Figure ‎4.25- The relationship between Ɗ and variation in MDSBA. 133](#_Toc508099301)

[Figure ‎4.26- Processing time comparison for the three-anonymisation methods 134](#_Toc508099302)

[Figure ‎4.27- Processing time comparison of various datasets and sizes 135](#_Toc508099303)

[Figure ‎5.1- SAML communication steps between SP and FS 140](#_Toc508099304)

[Figure ‎5.2- XML response in SAML assertion sections, and attributes section modification 141](#_Toc508099305)

[Figure ‎5.3-Live data options for data owner interface 144](#_Toc508099306)

[Figure ‎5.4-Archived data options for data owner interface 144](#_Toc508099307)

[Figure ‎5.5- Steps to configure initial values for each dataset 145](#_Toc508099308)

[Figure ‎5.6- Organization-Roles-Groups mapping 148](#_Toc508099309)

[Figure ‎5.7- MDSBA three main services, core, initialiser and anonymiser 149](#_Toc508099310)

[Figure ‎5.8- Sequence Diagram for MDSBA framework. 150](#_Toc508099311)

[Figure ‎5.9-The core part of MDSBA framework 152](#_Toc508099312)

[Figure ‎5.10-The core part of MDSBA framework 152](#_Toc508099313)

[Figure ‎5.11-Interface to create G groups of Q-IDs, classes, and classValues 153](#_Toc508099314)

[Figure ‎5.12- Definder.xml file demonstration 156](#_Toc508099315)

[Figure ‎5.13- Sample of data\_id.xml file 157](#_Toc508099316)

[Figure ‎5.14- Created Pig script file by the initialiser service 159](#_Toc508099317)

[Figure ‎5.15- PHP sample for anonymiser service 161](#_Toc508099318)

[Figure ‎5.16- Initialiser and Anonymiser algorithms 163](#_Toc508099319)

[Figure ‎5.17- Algorithm for Anonymiser service 164](#_Toc508099320)

[Figure ‎5.18- Sample of purge.xml configuration file 167](#_Toc508099321)

[Figure ‎5.19- Stage zero of anonymisation to protect from Obvious Guess 171](#_Toc508099322)

[Figure ‎5.20- Organization A and delegated roles 173](#_Toc508099323)

[Figure ‎5.21- Tradeoff between D and AGUI No. in [] 175](#_Toc508099324)

[Figure ‎5.22- Single Q-ID group access and obvious guess 176](#_Toc508099325)

[Figure ‎6.1- Cumulative Frequency for Seer G(QID)1 disruption 187](#_Toc508099326)

[Figure ‎6.2- Regression calculation for Seer G(QID)1 disruption 190](#_Toc508099327)

[Figure ‎6.3- Groups mapping to business roles 193](#_Toc508099328)

[Figure ‎6.4- AGUI Num. for both roles in seer data 194](#_Toc508099329)

[Figure ‎6.5- AGUI Num. for both roles in adult data 194](#_Toc508099330)

[Figure ‎6.6- Part of ERD showing the relationship between patterns and G groups 198](#_Toc508099331)

[Figure ‎6.7-Algorithm for determining the best patterns 200](#_Toc508099332)

[‎7.1- Spark structure and jobs distribution 206](#_Toc508099333)

[‎7.2- Spark structure in master and workers 208](#_Toc508099334)

[‎7.3- Comparison between Hadoop and Spark in dealing with memory and disks 210](#_Toc508099335)

[‎7.4-Comparison between the anonymisation algorithms of MDSBA and TDS 216](#_Toc508099336)

[‎7.5- Anonymisation algorithms in Pig and Spark 221](#_Toc508099337)

[‎7.6- Comparison in process-time between Pig and Scala scripts with 3 workers 226](#_Toc508099338)

[‎7.7- Comparison in process-time between Pig and Scala scripts with 4 workers 227](#_Toc508099339)

[‎7.8- Failed tasks in Pig script are shown in a high processing time 228](#_Toc508099340)

[‎7.9- Process time between filter/group and group only 229](#_Toc508099341)

[‎7.10- Performance comparison between caching and non-caching data 231](#_Toc508099342)

# LIST OF TABLES

[Table ‎2.1-Reserved system memory 23](#_Toc508099343)

[Table ‎2.2- Minimum container size recommendations 23](#_Toc508099344)

[Table ‎2.3- Security attack using side information 27](#_Toc508099345)

[Table ‎2.4- Multiple anonymisation example 33](#_Toc508099346)

[Table ‎2.5-A Non-anonymised data sample Table 2.5-B Anonymised data sample 35](#_Toc508099347)

[Table ‎2.6- Obvious Guess causes the Homogeneity attack 35](#_Toc508099348)

[Table ‎2.7- Generalized tables GT 36](#_Toc508099349)

[Table ‎2.8- Comparison between Bottom-Up Generalization and Incognito in number of searched nodes 39](#_Toc508099350)

[Table ‎2.9- A. Original inpatient data Table 2.9-B.Anonymized inpatient data 40](#_Toc508099351)

[Table ‎2.10-Diverse impatient data 41](#_Toc508099352)

[Table ‎2.11- Compressed Patient table 42](#_Toc508099353)

[Table ‎2.12- The specialization starts with Education, the highest InfoGain 43](#_Toc508099354)

[Table ‎2.13- LKC-Privacy model (Original data) 44](#_Toc508099355)

[Table ‎2.14-Generalizing the previous data by (Job, Age) 44](#_Toc508099356)

[Table ‎2.15-A. Medical Data set Table 2.15-B. Anonymisation pattern 1 47](#_Toc508099357)

[Table ‎3.1- Four Q-ID groups chosen randomly from Adult dataset 58](#_Toc508099358)

[Table ‎3.2-A. Original Data Table 3.2-B. Masking with P ≤0.13 Table 3.2-C. Masking with P ≤0.5 66](#_Toc508099359)

[Table ‎3.3- Three Q-ID attributes example 78](#_Toc508099360)

[Table ‎3.4- Three Q-ID attributes example 81](#_Toc508099361)

[Table ‎3.5- Anonymisation for SSG1 81](#_Toc508099362)

[Table ‎3.6- Anonymisation for SSG1 81](#_Toc508099363)

[Table ‎3.7- The masking Pattern for some data types 83](#_Toc508099364)

[Table ‎4.1- Seer Cancer Data Q-IDs and probability 104](#_Toc508099365)

[Table ‎4.2- Seer Cancer Data Class (four sensitive values) 104](#_Toc508099366)

[Table ‎4.3- Illustrates the needed programs for each Q-ID group 107](#_Toc508099367)

[Table ‎4.4- Sample of Seer data 114](#_Toc508099368)

[Table ‎4.5- The original Adult data sample 119](#_Toc508099369)

[Table ‎4.6-A Adult data after generalising EDU Table 4.6-B Adult data after EDU specializing 120](#_Toc508099370)

[Table ‎4.7- Heart Disease Q-ID attributes, propabilities and the class values 125](#_Toc508099371)

[Table ‎4.8- Kasandr Q-ID attributes, propabilities, and the class values 125](#_Toc508099372)

[Table ‎4.9- Dataset of 4.6 GB split for both methods 127](#_Toc508099373)

[Table ‎4.10- Various Datasets of 4.6 GB split for MDSBA 127](#_Toc508099374)

[Table ‎4.11- Disruption example with Adult data 131](#_Toc508099375)

[Table ‎5.1- Illustration example of Properties data for roles and Q–ID groups 146](#_Toc508099376)

[Table ‎5.2- List of files used by the initialiser service 158](#_Toc508099377)

[Table ‎5.3- Log file for the purge process 165](#_Toc508099378)

[Table ‎5.4- Obvious Guess example 168](#_Toc508099379)

[Table ‎5.5- Part of Seer data with two Q-ID groups 169](#_Toc508099380)

[Table ‎6.1- Seer dataset with three G(QID) groups 187](#_Toc508099381)

[Table ‎6.2- Seer data / G(QID)1 results after line regression calculation 189](#_Toc508099382)

[Table ‎6.3- Seer data / G(QID)2 results after line regression calculation 191](#_Toc508099383)

[Table ‎6.4- Security levels options setup by data owners to decide the optimal k value 191](#_Toc508099384)

[Table ‎6.5- Adult dataset with three G(QID) groups 192](#_Toc508099385)

[Table ‎6.6-Pattern 1 of census data presented by G(QID) groups 196](#_Toc508099386)

[Table ‎6.7- Pattern 2 of census data 196](#_Toc508099387)

[Table ‎6.8- Matrix for choosing the best pattern 197](#_Toc508099388)

[Table ‎7.1- Round down to the nearest 5 example 218](#_Toc508099389)

[Table ‎7.2- Interval Range Rounding down to the nearest 5 example 218](#_Toc508099390)

[Table ‎7.3- Grouped and anonymised table 222](#_Toc508099391)

[Table ‎7.4- Illustrates the new state after ungrouping records 222](#_Toc508099392)

[Table ‎7.5- Passenger’s records 229](#_Toc508099393)

# LIST OF ACRONYMS

1. OLTP Online Transaction Processing scaling
2. OLAP Online Analytical Processing
3. TDS Top-Down Specialization
4. BUG Bottom-Up Generalization
5. Q-ID Quasi Identifier
6. MDSBA Multi-Dimensional Sensitivity-Based Anonymisation
7. HDFS Hadoop Distributed File System
8. UDF User-Defined Function
9. Ownership Level
10. ω Sensitivity Factor
11. ψ Sensitivity Level
12. Ø Obsolescence Value
13. τ Aging factor
14. ρ Aging Participation Percentage
15. SG Fully-equivalent Group
16. SSG Semi-equivalent Group
17. NG Non-equivalent group
18. Ɗ Disruption
19. YARN Yet Another Resource Negotiator
20. JVM Java Virtual Machine
21. DAG Directed Acyclic Graph
22. RDD Resilient Distributed Datasets
23. SAML Security Assertion Markup Language
24. FS Federation Service
25. SP Service Provider
26. AGUI Across Group Unique Identifiers
27. NoSQL Note only SQL
28. ILPG Information Loss per Privacy Gain
29. G(QID) aggregated Q-IDs, which contains one class and two to four Q-IDs
30. P(QIDi) probability of each Q-ID
31. KDC Key Distribution Centre sever
32. TGS Ticket Granting Service
33. AS Authentication Service
34. LDAP Lightweight Directory Access Protocol
35. CF Cumulative Frequency
36. y Regression Line for
37. *k* percentage value
38. ϒ number of created groups

# - INTRODUCTION

Big Data is a new technology trend has become a fact as a reason of the massive data growth in the recent past. Digital data can be obtained from a number of quantitative and qualitative data sources, including smart phones, radio frequency identification sensors, Internet of Things, driver trackers, smart watches, smart glasses, embeddable, video recordings, audio recordings, radar, navigation sequences, cheap storages, cloud services, social websites, tablets, and others. The International Data Corporation estimates that, on a world-wide basis, the total amount of digital data created and replicated each year has grown exponentially from 1 Zettabyte (1,000,000 Petabyte) in 2010 to 15 Zettabyte in 2017 [1].

There is no rigorous definition of big data. However, the term big data refers to the massive amount of digital information [2]. Two major specifications distinguish big data from the conventional data: Online Transaction Processing scaling (OLTP), and Online Analytical Processing (OLAP). OLTP presents the storing and retrieving, while OLAP presents data analytics [3]. These two features can be gained by using a distributed environment, where many computers process the data in a parallel time [4]. Big data needs to be stored, retrieved and analysed. Thus, data analytics is one part of big data processes. Data is beneficial when it is analysed, so users gain more information, and are able to understand the bigger picture of the business activities. Hence, the term data analytics is involved with the big data [5].

Data analysis has a spanning multiple disciplines [6]. Analytics term is becoming an essential part of Information Technology business such as; medical, financial, industrial, transportation, government intelligence, and more. Consider data analytics as a prominent tool to monetize business data. Medical organizations request medical data of patients, hospitals, tools and equipment to find the best method of improving their business, and developing medicine and tools. Commercial side is not the only part of data analytics. In 2009, the American centres for Disease Control and Prevention (CDC) has failed to track the H1N1 disease around the United States in a real-time. The disease was spreading everywhere, and threatening the public health. Collected information from patients are two to three weeks lag. Google analysed over 50 million common search terms for Americans, and accurately tracked the areas infected by the flu virus by what people searched for on the Internet [7]. Banks rely on data analytics to develop their customer relationship, mortgage management, risk assessments, and fraud inspection [8]. Data analysis manifests a new exploitation for the recent technology, which supports a real-time collaboration between customers and business. Numerous companies have established their businesses based on the collected data from the collaboration between customers and companies. Facebook, and Uber are one clear example, where customers interact recursively with the business provider. The applications provided by the smart phones induce such an interactivity [9]. New technologies such as Internet of Things, smart cities, machine learning, and others, rely on data analytics as a major data provider.

Data analytics aims to provide statistical information as a whole, while protecting the privacy of the individuals in the dataset. However, privacy attacks in data analytics is a major concern, which emerged a need for protection policies and algorithms. Hence, scientists proposed several privacy models to reduce the probable attacks against data, by presenting two categories of privacy models: interactive and non-interactive categories. Interactive models tend to hide the actual data, and provide statistical results instead. Data owners provide interactive interfaces, where queries are submitted through to obtain statistical summary results. Protecting against queries is accomplished by sanitization approaches. This approach is conducted by adding noise to the input parameters or to the output results. The perturbation is a small numerical value that can be calculated by Laplace or Gaussian equations. These privacy models are known as differential privacy models [6, 10-12]. In the non-interactive models, the data owner, publishes an anonymised copy of the collected data, termed as anonymisation or de-identification. Also, data owner removes some personal identifier attributes such as names, birthdates, and social security numbers [13-15]. However, other auxiliary details cannot be removed for statistical and scientific purposes. Information such as age, gender, postcode, marital status, and education are essential information in data analytics.

Homomorphic encryption is another type of the interactive models. Its concept is similar to the differential privacy model, but Homomorphic encryption is more secure, and users cannot access encrypted data. Three types of Homographic encryption are still being developed by researchers: partially Homomorphic (PHE), somewhat Homomorphic (SWHE), and fully Homomorphic encryption (FHE) schemes. In PHE, either multiplication or addition calculation can be operated at once, but not both. SWHE can support a limited number of addition and multiplication operations. Eventually, FHE sustains both addition and multiplication, and can compute any function [16].

Interactive models are highly secure for certain tasks and firms. However, users may find it difficult to create relevant queries, while they read from a black box. Users are unable to access the actual data, they can only view attributes description. This does not provide a wide range of flexibility on working with data groups, domains, and sub-domains. On the other hand, non-interactive models provide a complete anonymised version of data, where users have the opportunity to view data and rectify the appropriate query for obtaining the statistical results. Moreover, non-interactive models consider the background knowledge by attackers, when performing attribute linkage protection. This intuition is essential since the recent few years. Cloud services and social media play a very strong role in providing adversaries with precise background knowledge.

In non-interactive models, auxiliary information may provide personal re-identification for a certain extent. These identifiers may not gain 100% of re-identification, but a risk of predicting some data remains high. For example, knowing the patient age, gender, and postcode, may lead to uniquely identifying that patient with 87% [17]. These identifiers are known as Quasi Identifiers (Q-ID). A popular anonymity model, *k-anonymity,* was formally studied by Sweeney [17]. The model suggests an anonymisation for Q-ID, which tends to find a group of attributes that can identify some tuples in the database. The model hides the sensitive values by ensuring the equivalency between records with at least *k* times[13]. Two different techniques were developed to gain the *k-anonymity*: top-down specialization (TDS) and bottom-up generalisation (BUG). The first technique is based on walking through the taxonomy tree from the top towards the bottom, known as the Top-Down Specialization. The second technique constitutes of techniques that generalize data from the bottom of the taxonomy tree towards its top. These two technique aim to find equivalency in each data domain. Examples of BUG are proposed in *ℓ-diversity [18]*, and Incognito [19]*.* Example of TDS are proposed in *LKC-Model* [20], *(α, k)-Anonymity* [21], and the multi-dimensional TDS (MDTDS) [22] [23].

The previously mentioned anonymisation models were proposed for average size data. Big data manifests different scalable approaches, which makes anonymisation imposes alternative techniques. There was a need for more relevant models in order to cope with large sizes of data. Proposed models should consider big data processing tools of parallel distributed computing, such as MapReduce. However, Recent proposed models, such as parallel BUG [24], hybrid BUG / TDS [25], and Two-Phase TDS [26], are quite similar to the extant mentioned models for average size data. In fact, the modifications, over the previous versions, have degraded the information usefulness.

Moreover, there is no rigorous access control framework for big data analytics. The increased demand for big data analytics has promoted the publicity-driven business. As a result, a larger number of users from different firms are engaged to benefit from data analytics. This recalls a need for a large scale framework that is able to control users in a fine-grained access. The framework should be able to manage user’s authorization, and authentication. As mentioned earlier, anonymisation provides a complete version of anonymised data, which makes re-identification more probable. Currently, we are unable to assign the access permission for certain attributes. The needed framework should control permit/deny privileges on the data attribute level. This permits the access of the needed data only. Also, the framework should provide gradual levels of anonymisation as per user’s access privileges.

To fill the previously mentioned gaps, this research proposes a novel fine-grained access control framework. The framework follows the BUG anonymisation model, with a multi-dimensional sensitivity-based anonymisation (MDSBA). The framework provides a scalable anonymisation approach, with a parallel distributed computation computability technique.

## **Research Question**

As discussed in the introduction, big data suffers from lack of a robust framework that is able to manage access control for analytics. Moving big data to the cloud network emerged multi-tenants data storage, and multi-domain of user’s access levels. Security concept has been shifted toward a larger protection scale, where unknown number of users, organizations, and applications may access big data, and from anywhere and at any time. Concurrently, current provided anonymity solutions cannot be durable for such massive growing computational costs, and security threats. For these reasons, we may seamlessly lose control over the empowerment of data analytics. Resolving these concerns may utilize a comprehensive framework that is able to control access privileges in a fine-grained paradigm, mimics the role-based access control model, and supports scalability and performance of big data.

The following question is raised and derived from the Big Data concerns:

How a framework of Access Control Model can enforce the organizational business roles over Big Data analytics, with considering scalability and performance concerns. The framework:

* 1. Should participate in resolving privacy violation of data analytics in Big Data.
  2. Should enhance the efficiency of OLTP.
  3. Can resolve the big data granular access for analytics.
  4. Can enforce the external organizational policies by delegating them access and roles permissions.
  5. Suitable for multi-tenant and multi-domain environments.
  6. Establishes a fine-grained access control, by implementing data anonymity approach.
  7. Integrates RBAC concept within the same framework.

## **Thesis objective**

The core objective of this thesis is to provide a fine-grained access control framework for big data anonymisation. The anonymisation is provided by implementing *k-anonymity* base approach, which protects the re-identification of a person on accessing data for analytics. This is essential in multi-domain environment, where users of multi-access levels need to access big data for analytics. The framework provides security by insuring an escalated level of anonymity for less privileges users/firms, and a reduced level of anonymity for elevated privileges users/firms.

The framework leverages the granular anonymity, and able to control an effective access for a certain part of data. The role-based anonymisation control framework integrates organizational business roles on deciding the access permissions. Two access permission levels are provided; anonymisation level and number of attributes.

## **Thesis Contribution**

In this thesis, a novel multi-dimensional sensitivity-based anonymisation (MDSBA) framework is developed. The framework operates over the cloud network. The cloud network structure is a composition of data owner, a federation service (FS), and a service provider (SP). The framework is able to integrate the FS components and structure. On the other side of the cloud, SP accommodates multi-tenant data repository belonging to multiple owners. MDSBA enables data owner’s self-management for their own data. Moreover, organizations, who wish to participate in data analytics, are delegated to assign security privileges to their own users. Data owners protect their own authorization and authentication data on the FS side. Following this structure, the framework was divided into three main service; core, initialiser, and anonymiser. The core service operates on the FS side, which stores the details of: organizations, users, big data information, business roles, organizations security levels, anonymisation parameters, and Q-ID groups. The initialiser is located on the SP gateway side. Data owners upload big data parameters in XML format. The initialiser communicates with the core service throughout security assertion markup language (SAML) [27]. The method of the initialiser is to map between FS and SP access privileges and parameters. Finally, the anonymiser operates in the MapReduce domain, where NameNodes, and DataNodes servers apply the granular anonymisation by choosing one of the three masking methods: taxonomy-tree, interval, or suppression. The granular anonymisation is conducted by sensitivity equation that calculate the specific amount of applied anonymisation as per user’s access level.

The anonymisation granularity approach relies on two main factors: dynamic value of *k* in *k-anonymity*, and dividing data vertically into small number of Q-ID attributes. MDSBA calculates the user’s appropriate *k* value based on the data owner’s relationship closeness. Business co-owners have closer relationship than business partners or even public users. The closer relationship assigns a lower *k* anonymity value, which in turn a lower anonymisation level is applied. The anonymisation implements masking operations, by applying data probability on taxonomy-tree, interval or suppression. Data attributes consist of different data types, and each of which is anonymised by one or more of the masking operations. Hence, data with a taxonomy tree anonymisation nature can be generalized by moving from the root of the taxonomy tree toward the bottom. Numerical data anonymisation can be easily masked by applying an interval. The probability values is derived from the number of times that the attribute value may possibly appear. Eventually, the value of *k* participates in computing the probability value that each masking process should apply. The gradual anonymisation is controlled by the taxonomy tree generalisation level, the interval range, and the number of suppressed characters. More anonymised data imposes a lower taxonomy level, and a larger interval range.

The framework adopts Hadoop ecosystems, Pig, Hive, and Spark to operate anonymisation in highly scalable operations. MDSBA is constructed to fit Hadoop ecosystems operations. This is essential to avoid large size of data splitting into small blocks of data. Larger data size, as in big data, causes a computational data overflow on the temporary memory. Splitting data into small blocks of data size may degrade information gained. Thereby, an advantage of MapReduce features is gained to avoid such a data degradation. Ultimately, MapReduce adapts its own Hadoop distributed file system (HDFS). The file system is designed to read a large size of data blocks, around 128 MB. This structure revokes the need for splitting data in to small size of data blocks.

Moreover, this research contributes in the followings:

* The development of ownership levels and sensitivity levels equations. The ownership levels are affected by two factors of: ownership factor and time factor. The time factor is an optional value, which can be ignored by the data owner.
* The integration of an equation that is able to calculate the amount of anonymised data on any anonymised dataset, by using the disruption value (D). The D value calculates the masking value that was applied on each Q-ID attribute. Each anonymised data block is calculated, while the total value of anonymised blocks equals D.
* The development of a technique that is able to find the optimal solution of *k-anonymity* value. The solution provides data owners with few simple steps to calculate the most optimal *k* value.
* The comparison between MDSBA framework and other BUG and TDS models. The comparison includes performance, and level of information loss. Also, a comparison between Hadoop ecosystem tools was conducted.

## **Thesis Layout**

The remainder of this thesis is organised as follows:

**Chapter 2** introduces the background and challenges of this thesis. The chapter manifests the big data general definition and the difference between traditional data and big data. This shows the predominant difference is in data analytics. The chapter exhibits the big data analytics challenges, by comparing between data streaming and data batching processes. Moreover, big data analytics is prone for security threats and re-identification of a person. For this reason, the chapter explains amply several methods for preventing re-identification. All proposed security protection methods can be categorized under one of the three privacy concepts: differential privacy, Homomorphic encryption, and *k-anonymity*. Several methods were introduced as examples of differential privacy, and *k-anonymity*. After introducing security protection method for traditional data, similar sections are introduced for security protection methods for big data. The sections show why identical anonymisation methods for traditional data are not successful in big data.

**Chapter 3** preliminarily defines the multi-dimensional sensitivity-based anonymisation method (MDSBA), and then delves in the MDSBA details. The chapter starts by proposing the requirements for big data anonymisation methods, including: equivalency increase, focusing on security more than information gain, anonymisation algorithms should operate in a parallel environment, and gradual access. The rest of the chapter explores the MDSBA and the probability concept, which is the core part of the MDSBA method. Next sections describe data aggregation concept vertically and horizontally. Both probability and aggregation are the two main components of MDSBA structure. The last section before the summary describes the mathematical equations that are necessary to calculate the sensitivity level as per user’s access. This also includes the time factor and its impact on sensitivity value, and the masking operations in taxonomy tree and intervals. Moreover, the anonymisation algorithms were described in details for the groups.

**Chapter 4** illustrates the state of the art of MDSBA framework. The first sections illustrate MapReduce and Hadoop ecosystems. The next section presents Hadoop security, and the best protection methods for Hadoop network domain. After introducing Hadoop ecosystems and security, next sections apply MDSBA algorithms by using Pig Latin scripts. The scripts execute the anonymisation by applying several masking methods on taxonomy trees and discretization. The forth section compares between MDBSA and Multi-Dimensional Top-Down Specialization (MDTDS). The comparison includes some experiments that measure the prediction level through the classification error. The experiments prove that MDSBA is efficient in performance and has a very low prediction error in the large data size. The experiments were conducted on several datasets in both small and large data sizes. The last few sections focus on creating a new classification bench mark for measuring the performance of anonymisation methods. The sections introduce mathematical equations by implementing disruption concept to measure how much is the anonymisation impact on data.

**Chapter 5** explores the complete framework of MDSBA. The chapter starts by describing the communication method between the Federation Service (FS) and Service Provider (SP). The communication method is known by Security Assertion Markup Language (SAML). The second section proposes two types of datasets, archive and live data. Next sections present the details of access control and business roles. The sections describe the three services of MDSBA core. These services are distributed in FS and SP sides. The three services include: core, initialiser, and anonymiser. The sections conclude the method of generating the Pig Latin script. The second part of the chapter discusses two problematic concerns in MDSBA; Obvious Guess and Across Group Unique Identifiers (AGUI). The chapter part provides solutions for these two concerns. The solutions for Obvious Guess is established by creating a zero filtration stage before anonymisation, while resolving AGUI is established by increasing the value of *k*. The last section before the summary explains some experiments that measure the impact of the disruption values on AGUI.

**Chapter 6** discusses special topics regarding *k-anonymity* parameters. The chapter suggests a greedy-based heuristic approach toward an optimal *k* anonymisation value. This is a guidance for data owners on assigning *k-anonymity* parameters. The suggested proposal is related to the role-based anonymisation control framework. The framework provides a fine-grained access control by dividing Quasi-Identifier (Q-ID) attributes into vertical groups, with two to four attributes for each cluster. The chapter adopts two mathematical concepts to assign *k* value, which are cumulative frequency and linear regression. The linear regression requires more sophisticated calculations, so it outputs more rigorous results. The cumulative frequency is a special case of linear regression, when the high accuracy is not important. The section introduces *k* percentage parameter to manage and control the role-based anonymisation control. The last sections shed light on the dynamic groups of attributes.

**Chapter 7** engages another recent parallel distributed framework, known by Spark. The chapter compares between MapReduce and Spark in processing sensitivity-based anonymisation framework. The sections are divided in a sequence of Spark structure, general comparison between Spark and MapReduce, implementing MDSBA in Spark, a comparison between TDS and MDSBA, and finally a comparison between Spark in MapReduce in operating MDSBA. The section of implementing MDSBA in Spark describes the User-Defined Function (UDF) in Pig ecosystem of MapReduce, and in Spark. The section of “Implementing MDSBA in Spark” merely compares between UDF in top-down specialization algorithm, and UDF in MDSBA algorithm. The comparison shows a TDS intensive dependency on UDF through (IF statements) and iteration, while MDSBA implements UDF in much less dependency.

**Chapter 8** provides the conclusions and future work of this thesis. The chapter mainlydiscusses the way the research is developed throughout the end and highlights its contributions. It also reports on the limitations of this work. Finally, the potential future directions for this research are illustrated.

# - BACKGROUND AND CHALLENGES

This chapter introduces the background of this work. It first presents the general issues challenging the big data. The chapter is divided into four sections. The first section introduces general challenges in big data. The next section focuses on data analytics and research challenges. The third and fourth sections delve in the thesis contextual scope. They present various methods of protecting privacy in big data. The fourth section get a closer look at the *k-anonymity* methods and techniques. This chapter approaches to the thesis core study gradually, by starting from a general concepts and ending with a specific concept of data privacy protection.

Section two describes some challenges that face big data analytics. It focuses on general challenges such as the complexity of analytics algorithm, and the shortcoming of Pig and Spark frameworks. Both frameworks have limitations on iterated and conditional programming. Therefore, programmers are forced to use customized user-defined functions that are executed in black boxes. Customised functions cause several non-clear errors that may occur on executing big data. Spark and data streaming is a cumbersome in large data size. Deep analytics may cause memory overflow, which may unexpectedly terminate the program or the task.

Section three studies security in big data analytics. The major explained security challenge was the privacy violation, and personal re-identification. Various security breaches have been explained. Next, some popular solutions showed how we may prevent or reduce the personal re-identification attacks. Some of these popular prevention methods are: differential privacy and *k-anonymity*. In differential privacy, two software applications were described, Airavat and GUPT. Several challenges and impairments are presented in differential privacy. Another popular security prevention method is presented. *K-anonymity* was chosen over the differential privacy, encryption and Homomorphic encryption. Also, the *k-anonymity* impairments and challenges are presented in the same section. In section four, *k-anonymity* was presented with several methods of specialization and generalisation techniques in traditional and big data.

## Big Data Structure and Challenges

Establishing a proper big data structure is a complicated task. This complexity starts from establishing the proper infrastructure, which suits the available data type. It is a consequence decision, whereas choosing an infrastructure type may affect the future choice of other computing and application layers. The following explains how much it is sophisticated to pick up the right infrastructure, computing, and application layers.

The recent advancements of technology have led to a data transmission to the cloud network. The portable hardware devices, such as tablets, smart phones, and laptops, have urged investors to adopt the cloud infrastructure as an adequate solution. Software applications, were especially designed for these new devices, are technically correlated with the cloud. Big data term was coined to cope with the advancement trend. Big data exhibits unique characteristics if compared with traditional data. The unique characters are summarized in 3Vs of Volume, Velocity, and Variety. To deal with a massive size of date, we need efficient mechanisms to store, retrieve and analyse a large size of data. Hence, Volume refers to the massive size of stored data, with capability to scale up the storage size. Velocity is related to the performance and efficiency in handling data transmission and process. The transmission time describes the time spent for collecting or storing data among storage nodes within the cluster or across clusters. Variety is related to the variety of the data type of structured and unstructured. These big data characteristics call for new system architectures of data acquisition, transmission, storage, and large-scale data processing mechanisms [28].

Big data structure can be decomposed into three main layers, infrastructure, computation, and application. The infrastructure consists of pool of hardware devices, and device management systems. The virtualization system is part of this layer. This also include all applications related to network management and security. The second layer is the computation layer, which is a middleware between the infrastructure and the application layers. This layers is divided into three divisions; integration, management, and programming models. The integration is related data distribution and aggregation to/from data nodes within the cluster. This is presented by the file system. Many free source file systems were developed during the last decade, such as; [Quantcast File System](http://www.linuxlinks.com/article/20130411160837132/QuantcastFileSystem.html), Hadoop Distributed File System (HDFS), Ceph [29], Lustre, GlusterFS, Google File System (GFS), and PVFS. These proposed file systems were essential to replace the traditional network file systems. For instance, Network File System (NFS) is inefficient to handle very large data across many nodes. Moreover Storage Area Network (SAN) file system can be scaled-up, but it is extremely expensive as a reason of its dependency on fibre channels. The recent file systems were especially designed for big data. Most of them are provided with a parallel computation, and a map-reduce concept or a like. Also, they provide Portable Operating System Interface (POSIX). The network connections between the servers and the storage disks, such as NAS and NFS, are not recommended in MapReduce domain. Instead, the Direct Access Connection (DAS) is used, which is SCSI, SATA, or SAS. Eventually, many similarities in structure and operations are available in most big data file systems. The second division of management is related to big database management systems such as; NoSQL, SQL, and file systems. Finally, the third division is the programming model, which combines the management and file system together. It facilitates the data analysis applications. MapReduce [13], Dryad [42], Pregel [43], and Dremel [44] exemplify programming models.

The third layer is the application layer. This layer connects the application interface with the second layer of programming model. Both programming model and interface infer various data analysis functions such as; queries, statistical and classification. The application layer exploits the MapReduce and parallel distributed tools to query statistical analyses, machine learning and precision, classification and other analytics needs. The three consecutive layers are related to each other’s. Choosing the infrastructure should be considered based upon the application needs and functions. Choosing the proper file system is linked to the chosen programming model. Divisions and layers must be pre-planned before establishing the infrastructure. For instance, Hadoop Distributed File System (HDFS) does not efficiently operate with any storage virtualization like RAID, instead, I/O data is handled by Just Bunch of Drives (JBOD). Therefore, data storages should be provided with HDFS compatibility for the best performance. Another example, if an organization has decided to adopt NAS file system, then all storage devices must be provided with Fibre Channels [30].

In big data, it is essential to keep the three layers under continuous monitoring. The intensive amount of data may overwhelm the management level as a reason of choosing the inappropriate layer of infrastructure, management or application. This is a very sensitive and accurate choice, since an extra delay of microseconds may be accumulated exponentially with the continuous data increase. However, the complexity of the three layers, with verities of choices may mislead data owners on choosing the optimal solution for their applications. For management model, there are many kinds of databases. Even though, there is no optimal database for all data types. This depends on the workload scenario, speed of read / write, and many other options. Researchers have proposed general comparisons between database performance on read, write, latency, durability, synchronous, and asynchronous [31] [32].

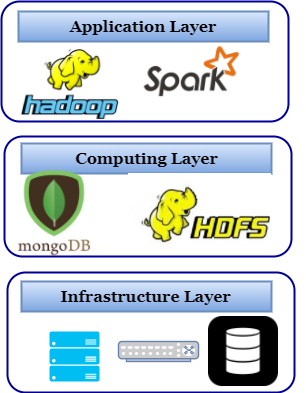


Figure ‎2.1- The three layers of big data structure

Researchers inferred that there is no single databases can make distinctive performance. Data of key-value, column, or document may perform differently with the varieties of database management systems. Also, database management models are different; some data operate on RAM, and keep a replication or a snapshot on disks. Other databases operate on disks and keep a cache copy on RAM. Also, some databases structure keep a high level of consistency and a low level of availability, or vice versa. Many NoSQL databases are available in the market, MongoDB [33] and simpleDB [34] for file data, Bigtable and HBase for Columns data, Dynamo and Redis for Key-Value, and PNUTS for rows data. Eventually, choosing the proper database type is one of the challenges in big data.

In a similar concept to database management model, programming model also contains different options. Data owners need to choose between; batch processing model, graph processing model, and stream processing model. The batch processing model deals with a snapshot of the targeted database. The stream processing model handles the real-time data, so the amount of copied data from disks to memory is very small. The third model is the graph processing model. This model suits some application, such as social media, where entities are related to one another [35]. This model nature is iterative, and the same dataset is revised many times. The most popular graph processing model is the Pregel [43] and GraphLab.

Batch processing model consists of two user-defined functions, map and reduce, known as MapReduce operations. Their concept is expressed by performing data intensive computations in parallel distributed operations. A MapReduce reads input files 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 outputs a list of key/value pairs. In the next phase, reducers combine the values belonging to each distinct key according to some functions and write the result into an output. The framework ensures fault-tolerant execution of mappers and reducers while scheduling them in parallel on any machine (node) in the system [36]. There are many batch processing models available in the IT industry, Hadoop and its ecosystems, and Dryad. Hadoop is a free-source framework developed by apache. Hadoop ecosystems resides at the top of Hadoop operations. An example of ecosystems are; Pig, Hive, and Spark.

MapReduce models provide a range of varieties for business needs. Some MapReduce models operate in memory, while others operate in disks. Pig and Hive operate in disks, which reduce the efficiency of iterative and interactive jobs. This imposes a continuous reading from node’s disks on each MapReduce operation. Moreover, each set of iterative operations (query) is counted as a separate MapReduce job, which incurs a significant latency [37]. In spark, the concept is different, since it implements the resilient distributed dataset (RDD), which represents a read-only collection of objects partitioned across a set of machines. The RDD is explicitly cached in memory across nodes and reused in multiple MapReduce-like parallel operations. This creates a temporary copy of the data from the disk the RAM, so all iterative and interactive jobs are computed in RAM. This technique reduces latency, which is usually caused by travelling time spent on input and output with the disk.

The three big data layers include a large number of technologies and models. Designing an adequate big data network is not an easy task. The hard part is finding the most compatible design that suits the data type and structure. This makes it even harder when data contains verities of files, multimedia and database sets. This diversity may recursively appear in multi-tenants data structure. When some users have a massive size of files, while another user needs to deal with data stream project. This diversity has created a kind of complexity. These challenges urge researchers to find a proper platform that is able to deal with a dynamic stack of operations. The stack should be able to choose the best matrix performance based on the data type and structure.

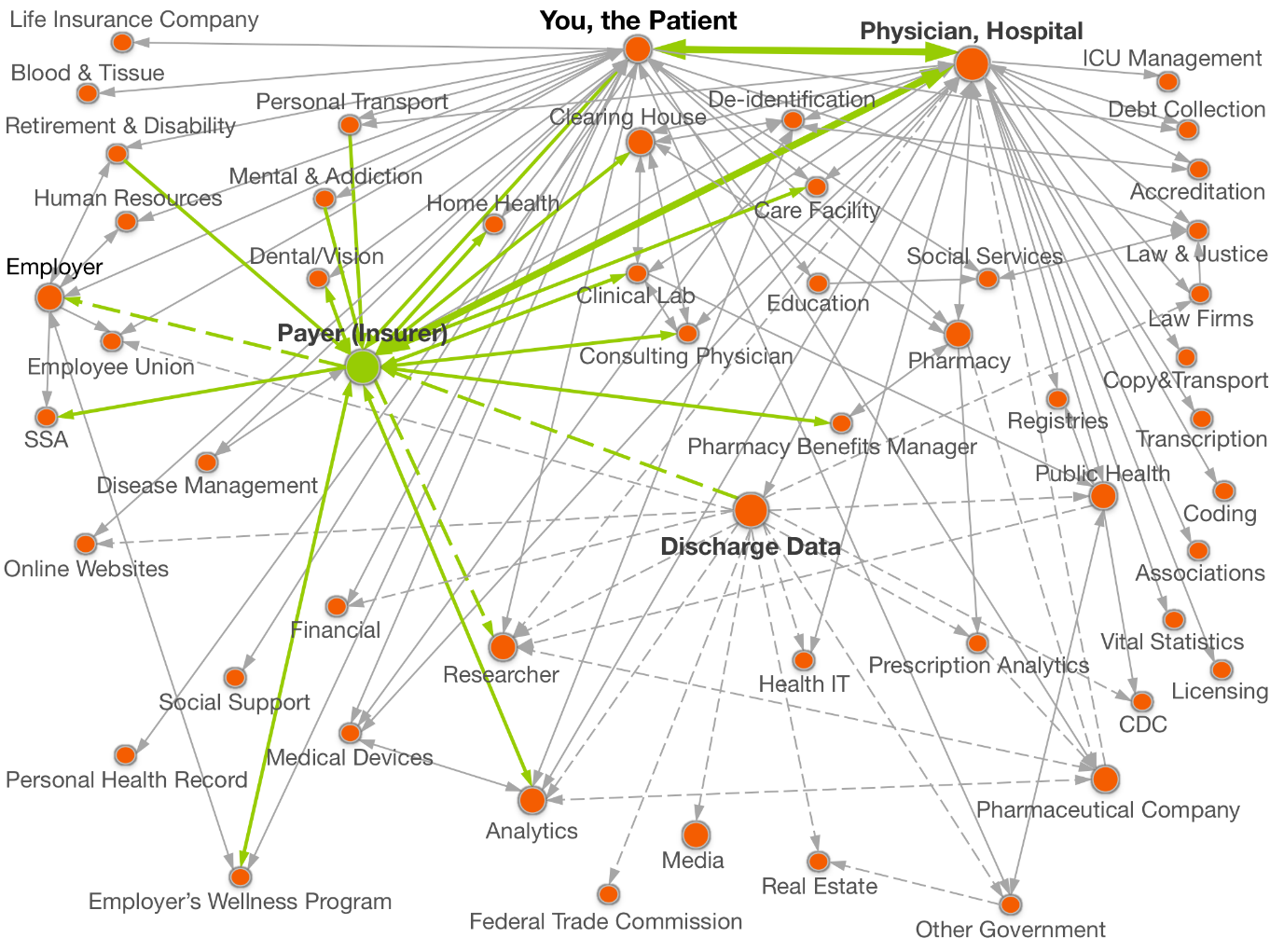
### Data Analytics

Data analytics is the most important part of big data. The aim is extracting useful information, which facilitates decision making, prediction, verifying the legitimate of data, or diagnosing and inferring faults reasons. The great diverse of data analytics methods and needs has derived several types of analytics categories. The categories can be summarized in descriptive, predictive, and prescriptive. The descriptive analytics implements data mining for insight analysis to find what has happened in the past. The predictive analytics implements statistics and forecast methods to predict the future behaviour. Eventually, the perspective analytics implements simulation to identify the system behaviour, and as a result the decision making [38].

Big data analytics has gained more popularity as a reason of the new technology trends. These trends included new business applications that rely on data analytics, network applications, and scientific application evolution. The earliest business data were intuitive and simple. Relational data bases management systems were able to accommodate and operate the available amount of data. General stored data were structural. The OLTP and OLAP were operated in a small scale. Since the beginning of the new century, there was a large shift in data collection techniques. The Internet has supported companies to provide some of their data online, which gives their own customers more interaction with their business, and better automation for their systems. [39]. A tremendous amount of products and customer information were offered by online participants. The clickstream data logs provided companies with an opportunity to study customer’s behaviour, and needed products and favourite services. Another wave of evolution has arisen after 2010, presented by smartphones. The number of sold smartphones and tablets exceeded the number of laptops and PCs. Portable devices and the Internet of Things created new features, such as location tracking, person-cantered care, and context awareness [40].

The new evolution of smartphones supported the development of new services, as a reason of the increased number of users. Most of these services were not possible few years back. Moreover, smartphones, technically, are known by non-pc computers, which imposes limitations in processing and storing data. The amount of data produced by individuals exceeds the capacity of these devices. This concern moved user’s data to the cloud network. Currently, most mobiles applications run on the cloud network. The majority global data is dominated by cloud network. The largest data size occupied by users is multimedia data such as images and videos. Social media has participated to a great extent in the data growth. It is estimated that more than 500 TB of data are uploaded on Facebook servers every day [41]. Moreover, scientific researches produce a huge volume of data from the fields of astrophysics and oceanography to genomics and environmental research. The National Science Foundation (NSF) has announced a BIGDATA project, which aims to advance the core scientific and technological means of managing, analysing, visualizing, and extracting useful information from large, and heterogeneous data to accelerate the progress of scientific discovery and innovation [42].

Medical data is one of the prominent data that have an intensive analysis demand. Patient’s data is precious for many parties and organizations. Medical data is rotated around and distributed to many medical and non-medical organizations. It is difficult to trace the medical data since the mesh network of transmitting information contains more than 50 different departments. Figure 2.2 illustrated the complexity of tracing patient’s data, and the critical need for such data. The figure shows one example of Insurers how they are able to access the Physician, hospital, patient, pharmacy, work, and many other locations with and without the patient’s name. This high demand on medical data recall a need for establishing a complete data access framework, with a fine grained access privileges.



Legend: http://thedatamap.org/legend-solid.jpg with your name,http://thedatamap.org/legend-dashed.jpg without your name

Figure ‎2.2- Data map shows the patient’s information distribution -source [43]

## Big Data analytics Challenges

One of the major challenges that face big data analytics is the analytics algorithm. Analytics technique in big data is unlike the traditional data analytics. Some mathematical, statistical, prediction methods, and simulation are similar in both traditional and big data. However, the technique applied to calculate and conclude results is different. In general, the un-structural data is mined and converted to structural data [44]. However, the volume of big data remains the main challenge in data analytics. Two major paradigms are expressed in big data analytics, batching and streaming analytics. Some applications may require fast and real-time analytics, while others not. Real-time analytics is needed in stock-trading analysis and alerts offered by financial services, fraud detection by examining transaction data, data and identity protection services, data generated by sensors and actuators embedded in physical objects, which is related to IoT, customer relationship management (CRM) applications, and clickstream analytics. In such applications, the processing time is essential, which shouldn’t exceed few milliseconds. The streaming process is continuous and infinitive, since the size of data is unknown. The infinite process fetches any new upcoming data, and proceeds this small portion to RAM. The fetching iteration is continuous, so the size of processed data is always small. The latest streaming frameworks are Storm, Flink, Kafka, and Spark [45].

In the batch-processing, data volume is known, the processing time is finite, and may last for seconds, minutes or even hours. Large data size is fetched from the storage, copied to RAM and processed. Therefore, large size of RAM and CPU is essential. MapReduce is the dominant model in batch processing. Data is divided into small chunks of data. The chunk are created by the file system and in parallel distributed manner in two phases, Map and reduce phases. This model schedules computation resources close to data location, which avoids the communication overhead of data transmission. The MapReduce model is widely applied in bioinformatics, web mining, census data, medical data, and machine learning. Depending on the application requirements, we may use streaming or batching mods. The differences in these two mods may encompass complex data storage and management systems, whereas in streaming mod, there is no data management system. One of the most popular MapReduce model is known by Hadoop. This model was especially designed for batch mode. Some models were especially designed to operate in the streaming mods, such as Storm, Samza, and Flink, while other models can operate in both mods, such as Spark [46].

Big data tools have gained a dramatic progress during the recent decade. Stream and batch tools are efficient enough to handle millions of data records within few seconds. Technologies like MapReduce have resolved the batch processing obstacles, while other technologies like Lambda architecture have resolved the stream and batch processing obstacles. Lambda technology consists of many frameworks such as Apache S4, Spark, Storm, Flink, and others. Some of these frameworks are dedicated for real-time and stream operations, while others can operate in both of stream and batch such as Spark. However, some operations in big data do not only need efficient frameworks, they also require efficient algorithms to benefit from the newly developed frameworks. One of the main parallel computing concerns is the algorithm structure. Ordinary programming and algorithms are inefficient in parallel processing. Special algorithms should be studied thoroughly to take the parallelism in to consideration. [47].

### Choosing the Proper Analytics Structure

Choosing the right analytics structure is one of the challenges that face data owners in big data analytics. Data owners need to choose the best computing framework that suit their data. There are several choices for; database management, computing framework, programming language, and others. For instance, data owners may choose a specific database management system like NoSQL base system. They may, also, choose Spark framework along with Spark SQL, and programmed by Scala. Figure 2.2 illustrates various number of choices for big data. For a better understanding, let us delve inside the structure of Apache Hadoop version 2.

MapReduce and ecosystems can run programs with the help of various languages such as; Java, Scala, Ruby, R, Python, and C++. Hadoop version 2 structure consists of two major divisions; Hadoop Distributed File System (HDFS), and Yet Another Resource Negotiator (YARN). MapReduce programs operate in parallel, so they can read and process a large data size at once. HDFS is a block structured distributed file system that can store petabytes of data within multiple nodes. Each block of data is replicated on at least three different nodes within the cluster. MapReduce network consists of a NameNode and DataNodes. The NameNode is the master, while DataNodes are the slaves. NameNode stores, manages, and serves the metadata of the file system. Hence, NameNode does not keep the file system table, instead, it just recognizes the data block availability and replication on which DataNode. DataNode manages and stores the actual data blocks as a per node service [48].

HDFS splits data into large chunk of blocks, 128 MB or larger. Therefore, the small size of files are not recommended in HDFS storage. DataNodes read data blocks from the HDFS, and transfer a copy of each block to parallel nodes for map processing. The reducer has three primary phases, shuffle, sort, and reduce. In the shuffle, the reducer copies the sorted output from each [Mapper](https://hadoop.apache.org/docs/r2.7.0/api/org/apache/hadoop/mapreduce/Mapper.html) using HTTP across the network. The sort phase sorts and merges the input by keys. The shuffle and sort phases occur simultaneously. HDFS operates with NoSQL database such as; HBase, and Casandra. Hive also supports the data management and collection as a warehouse. HDFS was developed especially for files management, therefore, HBase and other databases do not use all HDFS features and functions. For instance HBase is built at the top of HDFS, so HDFS process the data replication over nodes, by copying each block from the local server to the region server, which is a collection of data nodes. However, HBase is a dynamic database management system that enables many reads and writes features. Hence, HDFS is suitable for batch processes, while HBase is ideally suited for data stream [49].

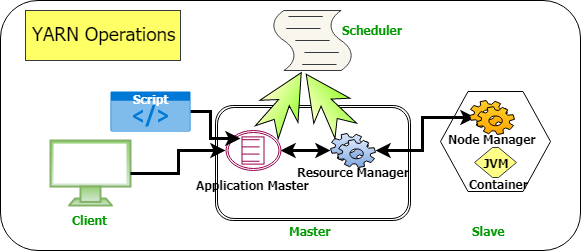


Figure ‎2.3-YARN structure in MapReduce

YARN is the new feature introduced in Hadoop v2. YARN is the architectural core of Hadoop that allows multiple data processing such as; interactive SQL, streaming, and batch processing. YARN is the foundation of the new generation of Hadoop, which enables organizations everywhere to realize a modern data architecture. It allows multiple distributed processing frameworks to effectively share the resources of a Hadoop cluster, as shown in Figure 2.3 and Figure 2.4.

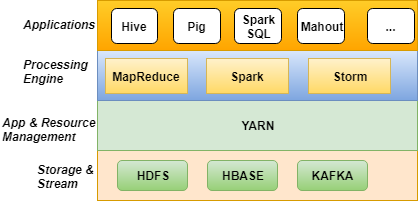


Figure ‎2.4- YARN supports an effective share of resources.

YARN manages nodes with the support of two main services, ResourceManager, and ApplicationMaster, as illustrated in Figure 2.3. The ResourceManager is interactively updated with the node resources and availability by the NodeManager. The node resources include: CPU, disk, and RAM. YARN is initiated when the user submits a job to Hive, Pig, or to any other models. ApplicationMaster is a per-application process that manages and coordinates the computations for a single application. Initially, ApplicationMaster is triggered by the user job submission. After job submission, the ApplicationMaster triggers the ResourceManager to obtain resources for the application. The ResourceManager deploys the NodeManager to find out the maximum available values of resource allocation. The availability is measured by a unit, called container. Each container of available resources has the right amount of CPU, disk, and RAM. Two main calculation operations are conducted by the NodeManager, *NUM\_OF\_CONTAINERS*, and *RAM\_PER\_CONTAINER*. The. This equation assigns the minimum value of the three resources to determine the number of containers. The. The reserved system memory is an essential term in both equations to calculate the total available RAM. Developers have suggested values of reserved RAM size as per total RAM size of each node. The suggested values are shown in Table 2.1. Table 2.2 shows the recommended MIN\_ CONTAINER\_SIZE. For instance, if the number of CPU cores in each node is 16, the number of disks in each node is 10, and the available memory is 48 GB, then NUM\_OF\_CONTAINERS =MIN(16\*2, 1.8\*10,(48-6)/2)=MIN(32,18,21)=18. The RAM\_PER\_CONTAINER=MAX(2,(48-6)/18)=MAX(2,2.3)=2.3≈2.

Once the ResourceManager defines the number of containers for the submitted job, the ApplicationMaster coordinates with the NodeManager to launch and monitor the application containers in the allocated resources. Leaving the coordination responsibilities to the ApplicationMaster will reduces the burden on the ResourceManager and will allow it to focus solely on managing the cluster resources. Moreover, creating an ApplicationMaster process for each separate submitted job improves the cluster scalability and performance [50].

Table ‎2.1-Reserved system memory

|  |  |
| --- | --- |
| **Total available RAM per Node** | **Reserved System Memory** |
| 4 GB | 1 GB |
| 8 GB | 2 GB |
| 24 GB | 4 GB |
| 48 GB | 6 GB |
| 64 GB | 8 GB |
| 256 GB | 32 GB |

Table ‎2.2- Minimum container size recommendations

|  |  |
| --- | --- |
| **Total RAM per Node** | **Recommended MIN\_CONTAINER\_SIZE** |
| Less than 4 GB | 256 MB |
| Between 4 GB and 8 GB | 512 MB |
| Between 8 GB and 24 GB | 1024 MB |
| Above 24 GB | 2048 MB |

Varieties of frameworks were developed for big data. MapReduce is one of the strongest base frameworks in big data processing. However, MapReduce management tools were away from the traditional DBMS, hence, there was a need to develop advance tools that are able to mimic traditional data tools. In the first version of Hadoop, there was no supportive tools to deal with structural data, such as SQL. In the second version of Hadoop, YARN was developed as a cluster middleware for several frameworks. Many MapReduce ecosystems have been released since onward, such as Pig, Hive, mahout, Spark and others. Pig data-flow was developed to narrow the gap between SQL and MapReduce. It is a high-level platform for creating MapReduce program. Pig offers SQL-like data modification constructs, which can be assembled in an explicit dataflow and interleaved with custom MapReduce style functions. Pig programs consist of a sequence of commands that are compiled into sequences of MapReduce jobs. Pig is an open-source project administered by the Apache Software Foundation. Pig compiles dataflow programs by a language called Pig Latin [51].

In addition to Hadoop framework, other recent frameworks were developed based on Lambda architecture, such as; Spark, Storm, Flink, Samza, and others. Most of these frameworks can be considered as hadoop ecosystems. This is because they are able to run at the top of YARN. One of the popular batching, and streaming frameworks is Spark. It has many advantages over the Pig Latin that mitigates the latencies, and increases the performance. Pig interacts intensively with the disk, by reading from HDFS and writing the results back to HDFS. These in/out transmissions consume a considerable time. Unlike Spark, which implements Resilient Distributed Dataset (RDD). RDDs is the main distinguishing feature of Spark. Spark adopts Scala as a de-facto language. The key programming abstraction in Spark is RDDs, which are fault-tolerant collections of objects partitioned across a cluster that can be manipulated in parallel.

As noticed, choosing the proper technology for data analytics is not an easy task. Data owners need to find out if the analytics is applied on stream or batch data. The data nodes infrastructure availability like memory, processors, and disks is very essential on determining the right structure.

## Security Challenges in Big Data Analytics

So far the previous sections addressed some challenges that face the analytics term in big data. In this section, more security challenges will be addressed with various solutions to resolve these concerns. Data analytics is prone for several attacks, and this can be categorized into four main attacks: storage, computation, communication, and privacy. In this research the main focus is the security of analytics operations in big data. Other kind of external or surrounding attacks are beyond the scope of this research. Miners, who attempt to access some datasets for analytics purposes, are prone to the four previously mentioned attacks [52]. Network administrator need to protect the network resources and operations of analytics. Next, the four main attacks are explained in details.

Storage attack may occur on any data node. Usually big data are stored in multiple nodes and replicated on a number of 3X nodes. In MapReduce frameworks, data is either stored as files, or structured in a database management solution. It was previously mentioned that big data management tools are non-relational, and they contain a close structure to the file storage. Regardless the data format, data can be encrypted on multi-level, starting from the disk level to the data set level or even cell level. Big data encryption methods are very similar to the traditional data encryption, such as; transparent, column-level, field-level, file system, and hashing [52]. Unfortunately, storing data with such levels of encryption will degrade the performance as a reason of the high computation cost on decrypting data before being analysed. Thereby, Homomorphic encryption was a proper solution, so miners can retrieve statistical results without unencrypting data. However, this type of encryption is immature yet, and still under research and development [16].

In computation and communication, MapReduce environment should be secure enough to handle miner’s analytics queries. Section 2.2 has already delved inside MapReduce framework. In Hadoop, the domain must be configured to switch to a secure mode. This includes processes and HDFS storages. Securing communication is directly related to securing computation. One of the major security challenges in big data analytics is the privacy re-identification attack. Medical, census, scientific, and commercial data may contain private details about individuals, who do not wish to share such information publically. The re-identification may occur even with hiding some attributes and values. In every dataset, there may be some sensitive information, such as medical status, should not be exposed to public. Researchers proposed different privacy protection techniques, by hiding all or part of the datasets. Each technique may best suit some research fields and needs in specific domains. Following, two privacy attacks and proposed methods for protections are presented.

In privacy attack, the re-identification may occur by three types of attacks: state attack, privacy attack, and timing attack. The state attack can be triggered by the adversary code, which may change the values of statistical variable, such as the keyword. In this case the privacy algorithms may lose the protection control. The attacker may run malicious code to transfer the other mapper’s output through the network. Another popular attack is the privacy, when the adversary reads some data and compares it with his/her external data. It is not necessary for the adversary to reach the sensitive data, it can be predicted based on the other attributes. The privacy attack may occur by side-link attack, or just guessing some private information as per homogeneity or background knowledge. Sometimes, users know some specific person’s private information, hence, guessing other sensitive information is not a hard task. Finally, the timing attack is possible by using an infinitive loop in the script, or by forcing scripts to run longer than the expected time. The time attack can also occur by the adversary using timing channel attack. The user keyword is also a prone for attacks [10] [11] [53].

Resolving privacy attack can be implemented by several security methods. The most popular methods are differential privacy and *k-anonymity*. The next few sections describe both security methods.

### Protecting Privacy by Differential Privacy

The main challenge in big data analytics is the need for external users to access data. Data is given in interactive or non-interactive forms. The interactive forms mandate users to abstract statistical summary results without giving the actual data. This form conceals the actual data from users. Instead of showing data, attributes descriptions are given to allow users creating their own queries. In non-interactive forms, users are given anonymised data for security and privacy protection. The interactive form can be applied by encrypted or plain data. Users submit queries, the system completes the statistical calculations, and returns the statistical results to the user. However, this form of results provokes a kind of privacy attack. To illustrate the possible security breach, let us give the following example; Table 2.3 shows a list of patients, with attributes of age, gender, name, and diabetes status, where (Has Diabetes=1) means positive. If the miner (analyser) has submitted query Q1=”total number of Has Diabetes =1”, while the second query Q2=”find the total number of Has Diabetes except Karen”. Since the user knows Karen’s name, and the abstraction of both queries Q2-Q1= 1, then Karen must have diabetes. This kind of attacks is very possible, even with hiding all data [10].

Table ‎2.3- Security attack using side information

|  |  |  |  |
| --- | --- | --- | --- |
| **Patient\_Age** | **Gender** | **Name** | **Has Diabetes** |
| 45 | Female | Marry | 1 |
| 40 | Male | Paul | 1 |
| 38 | Male | Mark | 0 |
| 55 | Female | Karen | 1 |
| 62 | Female | Nicole | 0 |
| 41 | Male | Steven | 1 |

Resolving such an attack is possible by applying differential privacy model [6, 10-12]. The model aims to eliminate some personal attacks, by adding noise to the input parameters or to the output results. The perturbation is a small numerical value that can be calculated by Laplace or Gaussian equations. Differential privacy are best defined as; the outcome of any analysis is, essentially equally likely, independent of whether any individual person joins or refrains from joining the dataset. In the previous example, a probability value can be added to the total number of patients with diabetes, so the results of the probability or noise value will be; P[have diabetes]=0.1, and P[have diabetes except Karen]=0.9 This protects the privacy re-identification of Karen, since the total of all patents have diabetes= 4.1, while the total of all patients have diabetes except Karen= 4.5. The privacy loss of each query is denoted by. This concept is mathematically stated as;

(‎2.1)

Differential Privacy was derived from two main thoughts. Firstly, the privacy method supposes that miners do not need to view data to retain any visible records. Secondly, increasing the statistical analytics queries may ruin the privacy and increase the re-identification probability. Hence, conducting deep analytics and creating extra analytics quires as per results output, may be contaminated by this privacy technique. In medical research, miners need to create queries as you go along with the output results. For instance, they may search for Anthrax symptoms in a specific region, while the output results show all symptoms similar to cold and flu. These results are inaccurate, and further queries are needed to filter out some common symptoms. Moreover, the added values of noise are generated automatically, which may leave a gap for queries manipulation by miners. One of the query manipulation method is choosing other known auxiliary data to mislead the system’s query recognition. Detecting queries manipulation depends on the algorithm used for detecting queries, and before applying differential privacy [54].

#### Differential Privacy Frameworks

Various differential privacy frameworks have been developed recently. The frameworks were specially developed for big data MapReduce operations. Software engineers still ongoing of developing differential privacy in real-life applications. Enterprise companies, such as Apple, have implemented differential privacy in their big data analytics [55]. The most popular software applications are PINQ, Airavat, and GUPT.

##### Airavat

Airavat is a novel MapReduce security and privacy framework. It provides Mandatory Access Control (MAC) for Mappers, by enforcing MAC on both sides of MapReduce processes and Mapper output. Airavat follows the Analytics process step by step starting from the user’s Mapper query, to determine if the required keyword is single or multiple. MAC policy is enforced during the MapReduce process. Finally, noise is added to the keyword, so the keyword can be compared with the key output. Airavat allows the execution of trusted and untrusted MapReduce on sensitive data, by enforcing the data owner policy using “Declassify Tag”. The latest versions of Airavat derived new security features such as; different side-channel attacks prevention, like state attack, privacy attack and time attack [56].

The architecture divides the procedures between three parties: the data owner, the user or mapper, and the computation framework (Airavat). The user first plans his/her code for Map and Reduce. Two types of users are pre-defined in Airavat, trusted and untrusted. The untrusted user’s keyword is obfuscated by noise on output. Also, the untrusted user is prohibited from executing all queries, queries like “list all” is not permitted. In contrast, trusted user is permitted to use any queries. Airavat cannot confine the keyword as a sensitive value or not, since it is unable to determine the string related attributes provided. It uses differential privacy to create noise by using Laplace equations. Moreover, it sorts trusted user keys prior the output, so they do not output in the same input sequence. As a result, the attacker cannot use the key order to leak information [11].

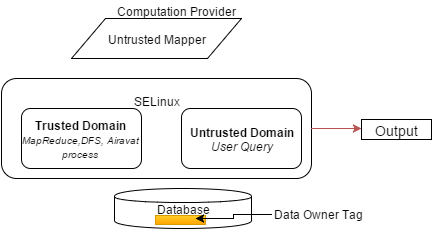


Figure ‎2.5- High-Level architecture of Airavat

SELinux is divided into two domains, one is untrusted for user code, and the other is trusted for: Airavat files, MapReduce, and DFS files. As shown in Figure 2.6, both of these domains are available in SELinux environment. The domain access and processes are controlled by MAC.

Airavat suffer from some limitations in confining the untrusted code, this is because of the keyword mapping difficulties, as described before. MAC is only used to control the user’s access and processes, while MAC does not provide any mechanism for interacting with differential privacy or choosing sensitive attributes. The access control is not implemented to fully distinguish different levels of access, hence, the user is categorized as trusted or untrusted, which is two levels of privileges [57].

##### GUPT

GUPT was proposed to reduce the analytics complexity for an average programmer. GUPT considers that the data owner and the service provider are trusted, while the analyst is untrusted. To gain this aim, GUPT framework is divided into three blocks: dataset manager, computation manager, and isolation execution chambers. Dataset manager is a database that maintains the data privacy, while computation manager handles the computations process by transferring data from the dataset manager, to the appropriate instances. Finally, the isolation execution chambers isolate and prevent any malicious behaviour. GUPT uses the differential privacy to protect the final output. It also uses the Laplace to explore the noise level and accuracy. The analyser query is evaluated on a smaller data blocks, since the search method implements small blocks of data and aging sensitivity. This means, the older age data are assigned by less sensitive values [11].

The optimal block size can be attained by finding a trade-off between the error and noise, which reduces the final error to a large extent. The block size varies from one query to another. This can be presented by , where *n* is the size of dataset, and is a parameter to be ascertained. Similar to Airavat, GUPT uses MAC to secure communication between instances. Each process performs in its designated space. AppArmor is used with SELinux as a sandbox to manage MAC [58]. The computation manager is split into server-client, the client allows GUPT to disable all network activates for the untrusted computation [11].

#### Possible attacks in Differential Privacy

Differential privacy is prone to several covert-channel attacks, which includes state attack, time attack, and private budget attack. Several proposals have introduced some solutions for these attacks. Airavat and PINQ suggested two separate domains, where the trusted domain contains the actual database, and the untrusted domain is left for the untrusted users. The untrusted users initiate an untrusted query, and return the results through the network. The state and time attacks cannot be avoided if the user was able to reach the trusted domain, since these attacks require executing at least two programs, one is the query and the other is a program that may measure the execution time, the CPU speed, the power usage, and other parameters. The time attack depends on measuring the query time with and without one record. The time difference may determine the record status. The private budget attack may occur as a result of manipulating the differential privacy algorithm. The algorithm assigns equations to measure the user’s query budget. It can be expensive when it contains some private requests. For instance, when the query asks about specific person’s name, or personal details [53].

As explained earlier, most attacks channels are caused by the user’s query. Developers spend a considerable time to fetch the query and calculate the query costs. This is essential to apply the on the output results. Regardless, the strength of the differential privacy algorithm, it is impossible to secure all queries by predicting the actual cost for each query. There are unlimited number of queries that may manipulate or mislead the program. Moreover, queries can be initiated by more than one adversary. For instance, adversary 1 may initiate a query1, which contains all viewers who watch adult movies in X suburb, and there age fall between 30-40. If the number of viewers was little, then another user may obtain the adversary 1 one results to build another query accordingly. Adversaries can be a group of users who have certain agreements on querying from data. This may prove that preventing user’s queries and attacks is impossible, and there is always a chance of security breaches [59].

### Protecting Privacy by *K-anonymity*

It was previously mentioned that data analytics can be either interactive or non-interactive. It was also explained how interactive analytics can be protected by differential privacy method. In non-interactive analytics, data is modified and anonymised to thwart re-identification attacks by ensuring that no individual’s record is unique in the data. In non-interactive approach, miners can gain actual data view for analytics. This form gives more powerful tools to deeply analyse data with unlimited number of queries. The first anonymisation method, known by *k*-anonymous, was proposed by Sweeney on 1998 [60]. More researchers have presented various methods related to *k-anonymity* concept.

One of the privacy techniques is the Quasi-Identifier (Q-ID), it implies finding a group of attributes that can identify other tuples in the database. These identifiers may not gain 100% of data, but even though, a risk of predicting some data remains high. For example, knowing the patient’s age, gender, and postcode, may lead to uniquely identifying that patient with 87%. Q-ID was implemented in *k-anonymity* method, and adopted as a scale for equivalency measurement. Only Q-ID attributes are verified for equivalency when investigating *k* value. However, Q-ID is a group of attributes chosen by data owners. So far, there is no clear technique to follow on assigning the Q-ID’s. Data owners choose a group of attributes, exposing these attributes together may thwart re-identification. Moreover, other auxiliary attributes may also support re-identification, therefore, Q-ID concept requires further study [61].

*K-anonymity* adopts the Q-ID definitions. It guarantees a privacy on releasing any record by adhering each record to at least *k* individuals, and this is correct even if the released records are connected to external information. The table is called *k*-anonymous; if one tuple has Q-ID values, and at least *k* – 1 equivalent records have Q-ID values. This means, the equivalence group size on Q-ID is at least *k* [13]. The method is stated formally by defining any Q-ID table RT=(A1,…,An) is said to be *k-anonymity*, if each sequence of values in RT appears *k* times. The principle of this definition aggregates QIDs by domains. This implies attributes in the table TR, and each value in the table appears with a sequence of *k* occurrence [17].

#### Impairments in K-anonymity

As a new concept of privacy preservation, a considerable number of impairments were reported by researchers in regard to *k-anonymity*. It is essential to consider this on proposing any future work for privacy preserving frameworks. The following concerns were reported by a various number of studies:

* **Multiple queries and anonymity variations:**

Sweeney has addressed few possible security failures against *k-anonymity* method [13]. An adversary may submit multiple queries for analytics, then anonymisation is applied on different Q-ID attributes on each anonymisation query. Adversaries may request data several times with multi-queries, so the anonymisation process may apply anonymity on the first Q-ID attribute in the first trial, and on the second Q-ID attribute in the second trial. Hence, linkage chance between the two anonymised tables is high. Table 2.4 illustrates an example of multiple-query attack.

Table ‎2.4- Multiple anonymisation example

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **GT1** | |  | **GT2** | |
| **Race** | **Zip** |  | **Race** | **Zip** |
| Person | 2138 | Asian | 2130 |
| Person | 2139 | Asian | 2130 |
| Person | 2139 | Black | 2140 |
| Person | 2138 | Black | 2140 |

The above example shows the impact of unsorted match between first and second anonymisation. In GT1, the Q-ID attribute (Race) was anonymised, while in GT2, attribute Zip was anonymised. Hence, re-identifying such records is highly possible, by connecting GT1 with GT2. Hence, both attributes can be easily identified. Table 2.4 reveals all actual values of Race and Zip by using two queries. In this case, *k-anonymity* masking is vain.

* **Finding *k*-value in *k-anonymity*:**

It was theoretically proven that finding *k* value is NP-hard [62] [63] [64]. Researchers studied the problem of anonymising data by column suppression. They showed that this problem is NP-hard for *k*≥2. The complexity of this problem for *k*=2 remained open.

* **Curse of Dimensionality:**

The curse of dimensionality is described as the extra-ordinarily rapid growth in the dataset attributes as the number of personal identifications (or the dimension) increases. With the attributes increase, the cost of computation grows exponentially with the dimension increase, which makes the cost prohibitive for moderate or large values of the dimension [65]. Because of the recent increase of personal information amount, the curse of dimensionality becomes a real problem that may conclude a bottle neck on applying *k-anonymity*. A significant amount of work has been done on the privacy preservation concern of different types of data. Numerous models [66] [67] have been proposed for the privacy preservation. However, it has been proven that increasing the Q-ID attributes will make anonymisation difficult with dimensionality increase [68] [69].

The high-dimensional attributes lead to a larger number of personal attributes. Obviously, more personal attributes may increase the background attacks. The masking or perturbation added to one attribute need be increased parallel with the increase number of attributes. This imposes a higher amount of obfuscation and a lower information gained with the dimensionality increase. One of the suggested solutions to the curse of dimensionality is to find dependent personal attributes by implementing feature selection[70]. The feature selection can be used to determine the maximum dependent attributes, in order to reduce the dimensionality of the dataset and retain a small subset of attributes. However, reducing the dimensionality of datasets will negatively affect the significant amount of information gained. The feature selection process depends to attributes transformation, which inevitably impact the final statistical results. Another suggested solution to the curse of dimensionality is the concept of horizontal fragmentation. The idea is to break up the attributes into small subsets of attributes using horizontal fragmentation and anonymise each subset independently. The small anonymised subsets are then aggregated together. Even some attributes are anonymised, however, all attributes are retained. This method reduces the large information loss that may occur on applying feature selection method [71].

* **Background Knowledge:**

This kind of knowledge is one of the most manifested attacks. The attack may occur if the adversary has some background knowledge about the user, such as age, sex, address, nationality and others. For example, Table 2.5-B shows the data list that the adversary have, where Table 2.5-A shows data before anonymisation. The adversary (Alice) is able to view Table 2.5-B only. When Alice has a pen-friend named Umeko who is admitted to the same hospital as Bob, whose patient record also appears in the table shown in Table 2.5-B. Alice knows that Umeko is a 21 year-old Japanese female who currently lives in the zip code 13068. Based on this information, Alice learns that Umeko’s information is contained in record number 1, 2, 3, or 4. Without additional information, Alice is not sure whether Umeko was diagnosed with HIV or heart disease. However, it is well-known that Japanese have an extremely low incidence of heart disease, especially in this young age. Therefore, Alice concludes with near certainty that Umeko has an HIV [72].

Table ‎2.5-A Non-anonymised data sample Table 2.5-B Anonymised data sample

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Non-Sensitive** | | | **Class** |  | **Non-Sensitive** | | | **Class** |
| **Zip Code** | **Age** | **Nationality** | **Condition** |  | **Zip Code** | **Age** | **Nationality** | **Condition** |
| 13053 | 29 | Russian | Heart Disease |  | 130\*\* | <30 | \* | Heart Disease |
| 13068 | 28 | American | Heart Disease |  | 130\*\* | <30 | \* | Heart Disease |
| 13053 | 23 | Japanese | HIV |  | 130\*\* | <30 | \* | HIV |
| 13068 | 20 | Japanese | HIV |  | 130\*\* | <30 | \* | HIV |

* **Homogeneity Attack:**

This kind of attack may appear after completing the anonymisation process. If a group of anonymised records contain similar sensitive information, then the adversary can obviously guess the person’s condition. Table 2.6 shows a sample of anonymised data, with similar sensitive values of (Cancer). If the victim lives in the post code 2116, then he must have a cancer.

Table ‎2.6- Obvious Guess causes the Homogeneity attack

|  |  |  |  |
| --- | --- | --- | --- |
| **Non-sensitive** | | | **Sensitive** |
| **Postcode** | **Age** | **Nationality** | **Condition** |
| 21\*\* | 3\* | \* | Cancer |
| 21\*\* | 3\* | \* | Cancer |
| 21\*\* | 3\* | \* | Cancer |
| 21\*\* | 3\* | \* | Cancer |

## *K-anonymity* Frameworks

### *K-anonymity* Methods for Traditional Data

#### Generalization

The generalisation method was initially proposed by Sweeney. *K-anonymity* suggests a data generalisation and suppression for quasi-identifiers (Q-ID). The original *k-anonymity* method defines Minimum Generalization (MinGen), and Maximum Generalization (MaxGen). If the curator requests a query with two Q-ID attributes, then the MinGen can be represented by omitting some values, or replacing them. The MaxGen implies values suppression, or hiding them completely. In each domain of a table T, a Domain Generalization Hierarchy DGH for an Attribute A is defined within a tuple t(A). The generalisation g for the table T is defined as g(T). The generalisation level (z) depends on the attribute value (νi). The following relationship implies the existence of the Value Generalization Hierarchy VGH for any attribute A for a function (f).

(2)

The generalisation is defined as:

(3)

Some values can be generalized up to three levels before suppression has occurred like postcode, while other attributes are generalized to multiple z level, such as the taxonomy tree. The following examples illustrate the main concept of *k-anonymity*:

Postcode generalisation DGH(Postcode): *Z0(2100,2109,2175), Z1(210\*,217\*), Z2(21\*\*), Z3(\*\*\*\*).*

Race Generalization DGH(Race): *Z0(Anglo,South American,African), Z1(person), Z2(\*\*\*\*\*\*).*

The generalized tables results are: GT(1,0), GT(1,1), GT(0,2), GT(0,1), as show in tables 2.7. Notice that GT(3,2), GT(2,2) and others are not possible in generalisation, as they are assigned on suppression.

Table ‎2.7- Generalized tables GT

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Race (E0)** | **P.Code**  **(Z0)** |  | **Race (E1)** | **P.Code (Z0)** |  | **Race (E1)** | **P.Cod(Z1)** |  | **Race (E0)** | **Postcode**  **(Z2)** |  | **Race**  **(E0)** | **Postcode**  **(Z1)** |
| Anglo | 2100 |  | Person | 2100 |  | Person | 210\* |  | Anglo | 21\*\* |  | Anglo | 210\* |
| S.Amer | 2109 |  | Person | 2109 |  | Person | 210\* |  | S. Amer. | 21\*\* |  | S. Amer. | 210\* |
| African | 2100 |  | Person | 2100 |  | Person | 210\* |  | African | 21\*\* |  | African | 210\* |
| Anglo | 2175 |  | Person | 2175 |  | Person | 217\* |  | Anglo | 21\*\* |  | Anglo | 217\* |
| Anglo | 2109 |  | Person | 2109 |  | Person | 210\* |  | Anglo | 21\*\* |  | Anglo | 210\* |
| S.Amer | 2175 |  | Person | 2175 |  | Person | 217\* |  | S. Amer. | 21\*\* |  | S. Amer. | 217\* |
| African | 2100 |  | Person | 2100 |  | Person | 210\* |  | African | 21\*\* |  | African | 210\* |
| S.Amer | 2175 |  | Person | 2175 |  | Person | 217\* |  | S. Amer. | 21\*\* |  | S. Amer. | 217\* |

PT GT(1,0) GT(1,1) GT(0,2) GT(0,1)

The above tables can be distinguished by the precision value. The higher precision is the chosen generalisation option. The precision table is calculated using the following equation:

In Table 2.7, DGH(Postcode)=3, and DGH(Race)=2 The number of attributes PT=2, and the number of tuples NA=8. Hence, the above table’s precision results are:

Prec(GT(1,0)=1- 8/2/16 = 0.75

Prec(GT(1,1)=1- (8/3 + 8/2)/16 = 0.58

Prec(GT(0,1)=1- 8/3/18 = 0.83

Prec(GT(0,2)=1- 16/3/16 = 0.67

The above calculated values prove that the highest precision is GT(0,1)=0.83, therefore, it will be picked up by the generalisation algorithm [15].

The generalisation using PT for each attribute is practically not possible for a large size of data, therefor, the real-world data is generalized and suppressed using tuples instead of individual attributes. One of these systems that can be implemented in the real-world data is datafly system [60]. The system guarantees the *k-anonymity* results, but does not necessarily guarantee the MinGen of data distortion. However, datafly is not very accurate, its decision is crude, since it generalizes all values associated with an attribute and suppresses all values within a tuple. Datafly is given the most important field, so it will be generalized, for example D\_O\_B is generalized to the year of birth. The next step is counting the number of times of the tuple occurrences. The non-repeated tuples with frequency=0 will be suppressed. Another popular system is μ-Argus, this system categorizes the attributes based on their sensitivity. Four values are assigned to measure each attribute’s identifying level: 0 (Not Identifying), 1 (Most Identifying), 2 (More Identifying), and 3 (Identifying) respectively. μ-Argus suppresses cells instead of suppressing the whole tuple, as in datafly [13].

The generalisation algorithms apply similar concept in taxonomy trees and suppression. The concept endeavoured to gain he *k-anonymity* by generalising Q-ID attributes starting from the bottom of the tree, and moving upward at the top of the tree, known by Bottom-Up Generalization (BUG). Several algorithms and frameworks were proposed based on BUG concept. Here are two methods of Incognito, and *ℓ-diversity* as BUG examples.

##### Incognito

Incognito method is derived from *k-anonymity* model to limit the confidence of the implications from the quasi-identifier to a sensitive value. Incognito iterates the Q-ID attributes by generalising the lattice nodes using join and prune. The method implements an algorithm of breadth-first search, by iterating the global-records several times to compute the frequency set of persons. SQL select queries are applied with (GROUP BY) for each two Q-IDs. The frequency number is calculated for each Q-ID attribute and compared with the other Q-ID attributes. The concept is finding the Q-ID attributes that need to be anonymised before commencing the anonymisation process. Finding them is conducted by calculating the most frequent appearance of each Q-ID and then building the lattice accordingly. The lattice is similar to the taxonomy tree concept. Each Q-ID is given two numbers, one denotes its sequence and other denotes the taxonomy level. For instance, the taxonomy tree of zero level for three Q-IDs is presented by:

*L0 (Level Zero) = {Q-ID(0,0), Q-ID(1,0), Q-ID(2,0)}*

After computing the frequency set for each root, the lattice results may impose two options of generalisation. The first option moves one level up for the Q-ID0 taxonomy, one level up for the Q-ID1, and no generalisation for Q-ID2, as shown below:

*L0 – L1= {Q-ID(0,1), Q-ID(1,1), Q-ID(2,0)} [first option]*

The second option moves one level up for Q-ID0, no generalisation for Q-ID1, and two levels up for Q-ID2, as shown below:

*L0 –L2 = {Q-ID(0,1), Q-ID(1,0), Q-ID(2,2)} [second option]*

These value present the minimum anonymisation level for non-equivalent records. This is an initial calculation for generalising tuples globally. In the second calculation, the distance vector can be derived. The distance vector is found by subtracting the two domain vectors {DA1… DAn} and {DB1... DBn}, that is a vector DV = [*d1... dn*], where each value *di* denotes the length of the path between the domain DAi and the domain DBi in the domain generalisation hierarchy Hi. After calculating the distance victor, and determining the best lattice on generalisation, a full-domain generalisation is applied to all dataset [19].

This method gains better performance than the original BUG method of Sweeny and Samarati. The performance increases parallel with the increase number of Q-IDs, if compared with BUG. Table 2.8 compares the number of nodes searched between BUG and Incognito. However, several disclosures are inherited from the original *k-anonymity* method, such as the curse of dimensionality and background knowledge. Moreover, the full-domain generalisation reduces the information gained in traditional datasets. Thus, Incognito leverages performance and reduces information usefulness.

Table ‎2.8- Comparison between Bottom-Up Generalization and Incognito in number of searched nodes

|  |  |  |
| --- | --- | --- |
| **Q-ID Size** | **Bottom-Up** | **Incognito** |
| 3 | 14 | 14 |
| 4 | 47 | 35 |
| 5 | 206 | 103 |
| 6 | 680 | 246 |
| 7 | 2088 | 664 |
| 8 | 6366 | 1778 |
| 9 | 12818 | 4307 |

##### *ℓ-diversity*

The *ℓ-diversity* is introduced by Machanavajjhala et al [14]. This algorithm aims to reduce the attributes linkage by preventing attributes discloser. The k-anonymity protects against identity discloser, while it cannot prevent attribute discloser. The attribute discloser is identified by homogeneity attack and background knowledge attack. It is developed from the fact that a *q* block is *ℓ*-diverse if it contains at least *ℓ* “well-represented” values for the sensitive attribute S. The ℓ-diverse is calculated using the entropy, by grouping the Q-ID and calculating the entropy for the groups using the following:

(5)

For example, let us consider Table 2.9-A as a part of inpatient data, and Q-ID={Zip, Age, Nationality}, while the sensitive attribute is Disease. The records are grouped or compressed with the similar Q-IDs. Based on the above given definition, Table 2.9-B represents k-anonymity generalization, but it does not satisfy ℓ-diversity [73].

Table ‎2.9- A. Original inpatient data Table 2.9-B.Anonymized inpatient data

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Zip Code** | **Age** | **Nationality** | **Disease (Class)** |  |  | **Zip Code** | **Age** | **Nationality** | **Disease (Class)** |
| 1 | 13053 | 22 | American | HIV |  | 1 | 130\*\* | < 30 | \* | HIV |
| 2 | 13068 | 25 | European | HIV |  | 2 | 130\*\* | < 30 | \* | HIV |
| 3 | 13068 | 27 | Russian | Heart Disease |  | 3 | 130\*\* | < 30 | \* | Heart Disease |
| 4 | 13053 | 28 | African | Heart Disease |  | 4 | 130\*\* | < 30 | \* | Heart Disease |
| 5 | 14853 | 52 | Asian | Cancer |  | 5 | 1485\* | ≥ 40 | \* | Cancer |
| 6 | 14853 | 56 | Indian | Heart Disease |  | 6 | 1485\* | ≥ 40 | \* | Heart Disease |
| 7 | 14850 | 46 | Japanese | HIV |  | 7 | 1485\* | ≥ 40 | \* | HIV |
| 8 | 14850 | 48 | Russian | HIV |  | 8 | 1485\* | ≥ 40 | \* | HIV |
| 9 | 13053 | 33 | African | Cancer |  | 9 | 130\*\* | 3\* | \* | Cancer |
| 10 | 13053 | 36 | Asian | Cancer |  | 10 | 130\*\* | 3\* | \* | Cancer |
| 11 | 13078 | 39 | Indian | Cancer |  | 11 | 130\*\* | 3\* | \* | Cancer |
| 12 | 13078 | 38 | Japanese | Cancer |  | 12 | 130\*\* | 3\* | \* | Cancer |

Table 2.9-B shows that records 9 -12 are prone to homogeneity attack. This imposes a need for another method of anonymization. The new method should unintentionally increase the generalization level, and as a result reduce the information gained. The entropy equation is used to re-calculate the entropy for smaller blocks of data. If data block Q is split into two sub-blocks Qa and Qb then entropy (Q) ≥ min (entropy (Qa), entropy (Qb)). This imposes that in order for entropy ℓ-diversity to be possible, the entropy of the entire table must be at least log(ℓ) [74].

The recursive (c, ℓ) diversity is implemented through finding another way of aggregating attributes to re-group the records as shown in Figure 4.10. It is clear that the Age anonymization was intensively generalized to include a larger age interval. The table shows tuples shuffling caused by the re-grouping. This kind of sequence supports the elimination of homogeneity that appeared in Table 2.9-B.

Table ‎2.10-Diverse impatient data

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Zip Code** | **Age** | **Nationality** | **Disease (Class)** |
| 1 | 130\*\* | ≤ 40 | \* | HIV |
| 4 | 130\*\* | ≤ 40 | \* | Heart Disease |
| 9 | 130\*\* | ≤ 40 | \* | Cancer |
| 10 | 130\*\* | ≤ 40 | \* | Cancer |
| 5 | 1485\* | > 40 | \* | Cancer |
| 6 | 1485\* | > 40 | \* | Heart Disease |
| 7 | 1485\* | > 40 | \* | HIV |
| 8 | 1485\* | > 40 | \* | HIV |
| 2 | 130\*\* | ≤ 40 | \* | HIV |
| 3 | 130\*\* | ≤ 40 | \* | Heart Disease |
| 11 | 130\*\* | ≤ 40 | \* | Cancer |
| 12 | 130\*\* | ≤ 40 | \* | Cancer |

However, several drawbacks can be found in this method. How about if the majority of the class values are limited by one type of disease. Also, this kind of re-grouping may cause a large loss in information gain. Moreover, ℓ-Diversity may be difficult or unnecessary to achieve.

#### Specialization

TDS algorithm was developed to achieve LKC-Privacy on high-dimensional data, the algorithm is also called HDTDS [23]. The idea is starting from the most general value in the taxonomy tree, and then move to the bottom of the tree. The taxonomy tree is pre-established for each attribute. At first, all tuples are generalized to the topmost root of the taxonomy tree (any), this value suppresses any quasi-identifier. The taxonomy tree is the prominent value that provides the masking operation in any attribute Dj. The masking operation can be identified by the data owner’s taxonomy tree as a topmost tree parent node, and a child or a leaf nodes ν, written ν 🡪child (ν). A specialized Dj can be viewed as a cut of a tree denoted as cutj.

The taxonomy tree should be built in advance for each attribute. This technique implements the specialization for the Q-ID attributes as a masking method. Several methods were proposed based on TDS algorithm. However, all methods follow similar procedures in finding out the best score within the Q-ID attributes. The attribute with the best score will be specialized. The score equation is shown below.

(6)

The value 1 was added to the denominator to avoid division by zero. This equation doesn’t satisfy the form matric to capture the classification, therefore, Shannon’s equation is used for correctness.

Next, InfoGain and AnonyLoss are found to determine the best generalisation for each attribute. This depends on the Q-ID used in each analytics.

(7)

Where denotes the entropy of T(x). To find out the best score for a compressed table, let us consider Table 2.11, which is derived from Adult data with an extra attribute of “Education”. This attribute describes the patient’s education level starting from the year 8 to postgraduate studies. Herein the TDS model is used, and the compressed records must start from the root of taxonomy tree.

Table ‎2.11- Compressed Patient table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Education** | **Sex** | **Work\_hrs** | **Class** | **# of Records** |
| 10th | M | 40 | 20Y0N | 20 |
| 10th | M | 30 | 0Y4N | 4 |
| 9th | M | 30 | 0Y2N | 2 |
| 9th | F | 30 | 0Y4N | 4 |
| 9th | F | 40 | 0Y6N | 6 |
| 8th | F | 30 | 0Y2N | 2 |
| 8th | F | 40 | 0Y2N | 2 |
| **Total:** | | | **20Y20N** | **40** |

To calculate the InfoGain, and AnonyLoss for each attribute, let us start first from the topmost generalisation, which is ANY\_Edu, for the whole records in the table.

* QID={Eudcation,sex,work\_hrs}
* Number of Records=40
* E(T[ANY\_Edu])=
* E(T[8th])=
* E(T[9th])=
* E(T[10th])=
* InfoGain(ANY\_Edu)= E(T[ANY\_Edu]) – (
* **InfoGain(ANY\_Edu)=1-(0+0+24/40\*0.65)=0.6**

While the InfoGain for the sex is calculated as:

* E(T[ANY\_Sex])=
* E(T[M])=
* E(T[F])=
* **InfoGain(ANY\_Sex)=E([ANY\_Sex])- (**
* E(T[1-99))=
* E(T[1-40))=
* E(T[40-99))=
* **InfoGain([1-99))=0.39**

Table ‎2.12- The specialization starts with Education, the highest InfoGain

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Education** | **Sex** | **Work\_hrs** | **Class** | **# of Records** |
| 10th | ANY\_Sex | [1-99) | 20Y4N | 24 |
| 9th | ANY\_Sex | [1-99) | 0Y12N | 12 |
| 8th | ANY\_Sex | [1-99) | 0Y4N | 4 |

The highest InfoGain is (ANY\_Edu), so the specialization for education starts first as shown in Table 2.12. In contrast, AnonyLoss shows that “Sex” should be generalized first, as calculating AnonyLoss is completed by the following equation:

AnonyLoss (ANY\_Edu) = A (QID) – A (ANY\_Edu (QID))

The average AnonyLoss is usually calculated to find out the best generalisation and specialization for each attribute. The total results can be determine by calculating the score for each attribute, the score(v)=InfoGain / (InfoLoss + 1), for example, Score(ANY\_Edu)=0.0165, and Score(ANY\_Sex)=0.0183, and for [1-99)=0.0136. This manifests that the ANY\_Sex score is the highest.

##### LKC Privacy

LKC-privacy model can be applied for the multi-dimensional data, such as patient’s information. This method assumed that the original *k-anonymity* and its extended privacy models exaggerate the security risk, since they assume that an adversary could potentially use any of the Q-ID attributes as background knowledge attack. LKC-privacy assumes that it is very difficult for an adversary to acquire all personal details accurately to launch an attack. Thus, it is assumed that the adversary’s background knowledge is bounded by at most L pairs of location and timestamp that the victim has visited. General intuition of LKC-privacy insures that Q-ID with a length of L and sensitive value of S is not greater than Class C, the idea is grouping length of records L in the data object T, by at least *k* records [75].

The following example illustrates the LKC-Privacy. Suppose that the following Table 2.13, and the taxonomy tree in Figure 2.6, where L=2, K=2, and C=50% (Yes or No). Table 2.14 was generalized using Figure 2.6 taxonomy. Based on the given information, let us determine whether the generalisation in Table 2.14 is correct or not, in relate to LKC-Privacy model.

Table ‎2.13- LKC-Privacy model (Original data)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | *Quasi-identifier (QID)* | | | *Class* | *Sensitive* |
| **ID** | **Job** | **Sex** | **Age** | **Transfuse** | **Surgery** |
| 1 | Cleaner | F | 35 | Y | Appendicitis |
| 2 | Cashier | F | 31 | Y | Appendicitis |
| 3 | Teacher | M | 35 | N | Urology |
| 4 | Engineer | M | 27 | N | Urology |
| 5 | Plumber | M | 25 | Y | Vascular |
| 6 | Electrician | M | 29 | N | Vascular |

Table ‎2.14-Generalizing the previous data by (Job, Age)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | *Quasi-identifier (QID)* | | | *Class* | *Sensitive* |
| **ID** | **Job** | **Sex** | **Age** | **Transfuse** | **Surgery** |
| 1 | Non-Technical | F | 30-60 | Y | Appendicitis |
| 2 | Non-Technical | F | 30-60 | Y | Appendicitis |
| **3** | **Professional** | **M** | **30-60** | **N** | **Urology** |
| 4 | Professional | M | 1-30 | N | Urology |
| 5 | Technical | M | 1-30 | Y | Vascular |
| 6 | Technical | M | 1-30 | N | Vascular |

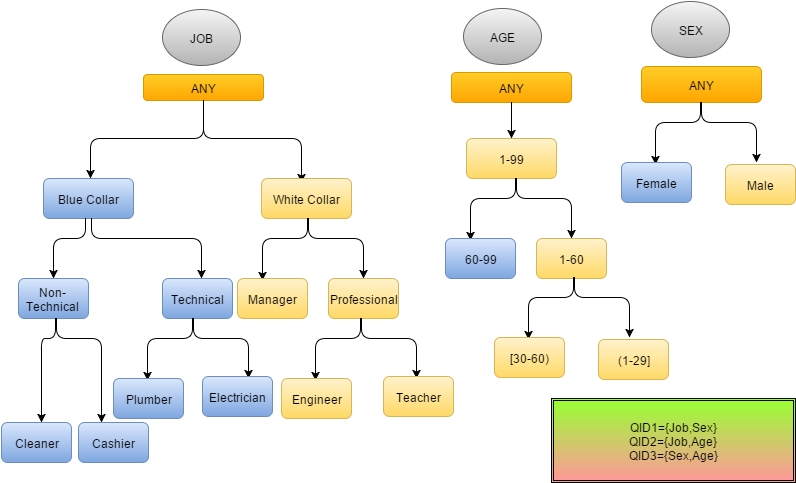


Figure ‎2.6- Taxonomy Trees for JOB, AGE, and SEX

As shown in Table 2.14, every two records can be grouped together. Only record #3 of {professional, M, 30-60} cannot be grouped with the similar record, as the age interval is different. This implies another generalisation level on age, for example between 0-90, which results higher utility loss [20].

This method can be suitable for dataset with larger number of *k* values. Small values of *k* may better get anonymised by using BUG. Moreover, several privacy re-identification attacks cannot be resolved by using this method. The next (*α,k)-anonymisation* is more effective [20]*.*

##### (*α, k*)-anonymisation

This anonymisation method is categorized under TDS methods. It was introduced by Wong et al. 2006 [21]. It can be considered as a special case of recursive ℓ-diversity. The method aims to protect both identifications and sensitive associations in a disclosed dataset. Two approaches are developed to prevent discloser, one is an extra security applied on some chosen sensitive values but not all. The second approach is an extension of Incognito anonymisation [19]. The first approach assigns a decimal value *α* to some sensitive values in the attribute (Class). For instance, if users were diagnosed with HIV, then the HIV value must be exclusively protected more than the other sensitive values such as, Flu, or Headache [76].

*(α,k)-anonymisation* is an extension of Incognito (eIncognito) with extra parameter α. The value of (*α*) denotes the de-association requirement for the protection, which requires that the proportion of each sensitive value in each group is at most α ∈ [0, 1]. The objective is to find a local recoding with a minimum cost, or with a minimum number of suppressed records. Incognito algorithm is an optimal algorithm for the *k-anonymity* problem. It has many advantages over the other anonymisation algorithms, such as resolving the diversity problem and making use of monotonicity property in searching the solution space. The search is a continuous iteration until finding the stopping condition. The stopping condition simply supposes that if a table T satisfies the privacy requirements, then every generalisation of the table T also satisfies the privacy requirement. The eIncognito applies Top-Down Specialization (TDS) operations, which is an opposite technique to bottom-up Incognito.

The method proved that implemented TDS in eIncognito reduces distortion by two to four times less. The generalisation or BUG imposes the use of global recording on anonymisation, which means one domain of generalization for the whole non-equivalent records in the dataset. In eIncognito, the use of TDS has reduced the distortion as a reason of replacing the local recording by the global recording. The local recording is shown in Table 2.15-C. To understand eIncognito in depth, we need to understand the EDGE PARTITION INTO 4-CLIQUES in the graph theory [77]. Based on this theory, the anonymisation cost is calculated for each record. For instance, if *k*=12, and α=0.5, we can interpret these two parameters by supposing that for 4 Q-IDs, there are 4 vertices in the 12 records corresponding to the edges in Q-IDs, then a cluster of these 12 records are formed where each modified record has four \*’s (suppression).

Let us study the following example to understand (α,*k*)-anonymisation. Table 2.15-A shows a subset of dataset with three Q-IDs and one sensitive attribute. Some of the sensitive attributes are more sensitive than the others. For instance, people who are diagnosed with HIV should be strictly protected from re-identifying their status. The other diseases, such as Flu and Fever are not serious so there is no need to strictly hiding them. Assume that *k*=2 and α=0.5. Inspecting the anonymised Table 2.15-B carefully, we can see that the anonymised table does not protect two patients’ with sensitive information of HIV infection. We may easily distinguish the two patients for the first two tuples if we know that one of them lives in the postcode 4350. Table 2.15-C is an appropriate solution. Since (\*, 1975, 4350) is linked to multiple diseases (i.e. HIV and fever) and (\*,\*, 4350) is also linked to multiple diseases (i.e. HIV and flu), which protects individual identifications and hides the implication. Table 2.15-C shows the eIncognito anonymisation using the local recording. The table shows high InfoGain if compared with Table 2.15-D, which is anonymised by the old Incognito method using the global recording. There are two goals for privacy preservation: (1) to protect individual identifications and (2) to protect sensitive relationships.

Table ‎2.15-A. Medical Data set Table 2.15-B. Anonymisation pattern 1

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Job** | **Birth** | **Postcode** | **Illness** |  | **Job** | **Birth** | **Postcode** | **Illness** |
| Cat1 | 1975 | 4350 | HIV |  | Cat1 | \* | 4350 | HIV |
| Cat1 | 1955 | 4350 | HIV |  | Cat1 | \* | 4350 | HIV |
| Cat1 | 1955 | 5432 | Flu |  | Cat1 | 1955 | 5432 | Flu |
| Cat1 | 1955 | 5432 | Fever |  | Cat1 | 1955 | 5432 | Fever |
| Cat2 | 1975 | 4350 | Flu |  | Cat2 | 1975 | 4350 | Flu |
| Cat2 | 1975 | 4350 | Fever |  | Cat2 | 1975 | 4350 | Fever |
| Table 2.15-C. Anonymisation pattern 2 | | | | Table 2.15-D. Global recording anonymisation | | | | |
| **Job** | **Birth** | **Postcode** | **Illness** |  | **Job** | **Birth** | **Postcode** | **Illness** |
| \* | 1975 | 4350 | HIV |  | \* | \* | 4350 | HIV |
| \* | \* | 4350 | HIV |  | \* | \* | 4350 | HIV |
| Cat1 | 1955 | 5432 | Flu |  | \* | \* | 5432 | Flu |
| Cat1 | 1955 | 5432 | Fever |  | \* | \* | 5432 | Fever |
| \* | \* | 4350 | Flu |  | \* | \* | 4350 | Flu |
| \* | 1975 | 4350 | Fever |  | \* | \* | 4350 | Fever |

### Critique of Traditional Data Anonymisation Methods

Traditional data is unlike big data. Traditional data comprises a limited number of data records. There is no threshold value to distinguish the traditional data from the big data. The rough data size and record’s numbers may provide a distinguisher between traditional and big data. Intuitively, we may consider few hundreds of thousands records are still traditional data. However, if anonymising data can be accomplished by a single machine in an acceptable time-manner, and does not need a parallel distributed operations, then data can be considered as traditional data. Anonymising traditional data algorithms do not require dataset split into small blocks. Also, the limited number of records reduces the operations failure and errors. This is because the small size of data can be smoothly uploaded and fit in the current server’s memories. For these two reasons, there is no need to rectify the current known anonymisation algorithms. The previously mentioned anonymisation methods in section 2.4 are enough and can accomplish operations in an acceptable time period.

Choosing the best anonymisation method is inaccurate. Some anonymisation methods may be reasonable for some datasets, but not for all. Different data records may require different anonymisation methods. Two main types of anonymisation methods can be chosen for various data types. In general, top-down specialization method is a suitable option when *k* value is quite large, while bottom-up generalisation is a suitable choice when *k* value is small. Determining the *k* value may depend on the data divergence and type. Two main reasons may urge data owners to choose TDS algorithms over BUG; the large number of attribute values, and the organization’s security policy. More secure anonymised data impose larger *k* values. For instance, if data owner noticed that the attribute EDUCATION contains many values with a wide range of education varieties, then he/she may decide TDS algorithm. In another example, if the organization’s security policy is low, then the *k* value can be smaller than 4, so BUG algorithm is a good option.

### *K-anonymity* Methods for Big Data

The previous BUG and TDS methods were also implemented in big data anonymisation. Few amendments are applied to suit the big data frameworks, in the matter of parallelization and distribution. The core concept of *k-anonymity* is similar to the previously mentioned methods. Similar techniques and algorithms are applied in both cases of TDS and BUG. Let us study some of these anonymisation methods to compare between the previously mentioned methods in traditional data and in big data.

#### Generalization

Several algorithms were proposed recently especially for anonymisation using MapReduce. Most BUG methods follow similar algorithm, by implementing BUG driver to leverage the information gain and security trade-off. The search metric computes the Information Loss per Privacy Gain (ILPG). These equations measure the entropy and the scores of each attribute. The algorithm generates a random number (ran). This number presents the number of random partition for the dataset (). Each sub-dataset is emitted to the MRBUG driver for an intermediate generalisation. This generalisation scan data, find the equivalent records < *k*, and merge Q-IDs up to Anonymisation Level one or two, that is AL1 or AL2. This intermediate generalisation is essential to reduce the final anonymisation computation. Finally, datasets are scanned again and the search metric computes ILPG again. For each sub-dataset, if < *k*, then find the best generalisation level and set to INACTIVE. Keep iterating and moving up the taxonomy tree, until *k-anonymity* is satisfied. As explained, the MRBUG driver operates twice, in intermediate and final. Firstly, it merges anonymisation, and secondly, it applies generalisation. This algorithm is found in [24, 78-80].

##### Advanced BUG

Pandilakshmi et al. [79] proposed Advanced BUG (adv-BUG). The Advanced BUG consists of the following steps: split data into smaller partitions, run the MRBUG Driver on partitioned dataset, combine the anonymisation levels of the partitioned dataset and apply generalisation to original dataset [79]. Other anonymisation methods use hybrid combination of BUG and TDS to anonymise data. A threshold value of *k* is determined by several algorithms to distinguish BUG from TDS use. The methods believe that BUG is more suitable for small *k* values, while TDS is more suitable for larger *k* values [25]. Some hybrid methods were recently proposed for big data by Zhang et al. and Irudayasamy et al. [24, 78, 81].

#### Specialization

Since the evolution of MapReduce and parallel processing, Roy et al. [56] presented a data privacy model, named Airavat . The system was developed after investigating MapReduce and differential privacy. This approach has encouraged researchers to re-design the available anonymisation methods for MapReduce computability. The TDS methods for big data were derived from the TDS proposed for traditional data. The miner rectifies have been contributed to the early versions of MapReduce framework. One of the predominant methods is known by Two-Phase Top-Down Specialization (TPTDS).

##### Two-Phase Top-Down Specialization

TPTDS depicts the two phase of Map and Reduce. The concept is very similar to the previously explained TDS, which depends on generalising all Q-ID attribute, and calculating the entropy and the score for each Q-ID. The highest Q-ID score will be specialized. This operation is iterated to find the best cut in the tree, or in the interval. In the first phase, dataset D is split into small chunk Di of data. Di denotes any block of data, where. The value denotes the number of blocks. An operation, known by MapReduce To-Down Specialization (*MRTDS)*, scans each data block in a subroutine in parallel to make full use of the job level parallelization of MapReduce. *MRTDS* driver is an intermediate anonymisation level that specializes data without violating *k* –anonymity. *MRTDS* driver is applied once in each phases. In the first phase, the driver provides some sub-datasets of a value, where. The term denotes the intermediate anonymity parameter, which is usually given by anonymisation experts. Formally, the *MRTDS* operates multi-tasks on each data block for initial specialization by. The anonymisation level presents the top generalisation level of the taxonomy tree, which is usually given by (any). Specializing Q-ID attributes is applied as per highest score attribute. Another program, known by Information Gain per Privacy Loss (IGPL). The IGPL calculates the highest score for each specialized Q-ID attribute. This technique is popular in most anonymisation operations and algorithms.

After completing the intermediate anonymisation, all (AL) values are aggregated and the next phase is initiated. The second phase operates *MRTDS* again to produce the best cut specialization. The algorithm is close similar to phase one algorithm. The second phase receives data from the intermediate output as per key-value of (key,list(count)). This phase updates the IGPL results that was initiated in the first phase. Initially, phase one lists all best specialization for each data block. In the second phase, the specialization is validated or updated with a new specialization value. The validation is accomplished by gaining two conditions: firstly, the parent value of specialization should not be a root, i.e. should not be (any). Secondly, the anonymity should be. Several iterations can find the best specialization cut for the chosen Q-ID. The IGPL updates the specialization list as per IG calculation, and the final list of specialization is updated and emitted, so the data records are masked with this list [82].

### Critique of Big Data Anonymisation Methods

Most big data anonymisation methods foster both TDS and BUG in a hybrid manner. The large data are managed and processed easier if hybrid technique was properly applied. This depends on the *k* value and other parameters. Both TDS and BUG were earlier used in traditional data, and there is no amendments have been done when implemented in big data. The concept of the Information Loss per Privacy Gain (ILPG) is the major core for all anonymised methods. This is true for traditional and big data. ILPG can be successfully implemented on a single machine, where ILPG driver iterates a large array of data records several times. The iteration attempts to find the best cut of generalisation or specialization. In big data, this algorithm can be a cumbersome for memory size and processor limitations. Thus, amending ILPG algorithm to fit the distributed system can be gained by splitting dataset into small chunk or data. This is the exact solution adopted by the current anonymisation methods. Chapter 3 proves that the data records equivalency increases parallel with the number of records increase. The positive fact of big data is the high percentage of records equivalency. Misusing this fact can negatively affect the information gain. We need a method that is able to benefit from this advantage, by splitting data in a nominal fashion, rather than conducting a random split. In the conclusion, the random split that is implemented in the current anonymisation methods is inadequate.

In addition to the random split disadvantage, the current big data tools operate differently. ILPG driver was developed based on the early releases of MapReduce. The latest MapReduce and Spark frameworks are operated in two level-stack. The first level is the core structure of the distributed system attached with the second level of an ecosystem. For instance, MapReduce consists of many ecosystems operate at the top of YARN, such as Pig, Hive, and HBase. For this reason, implementing anonymisation with the new released MapReduce ecosystems requires different algorithms. New released ecosystems diminished the iteration use, and limited the regular programming algorithms. For instance, Pig Latin script, Hive script, Scala script, and other programming scripts have limitations on using IF statements and iterations. These scripts were specially developed for parallel programming, and they are not flexible as the traditional programming. The reason is the scalability and performance concerns being considered on developing these programming scripts. If iterations and several IF statements are required, then UDF can take a part in programming the needed part of the code. The UDF can play a small and limited part of the code. The UDF is a black-box, as explained in chapter 4. Data needs to flow outside the parallel system to a stand-alone UDF. If the data flow was large, then there is no guarantee that the UDF will be able to handle this massive size of data. For this reason, data flowing to UDF must be limited and small. Unfortunately, ILPG algorithm depends completely on iteration and IF statements. The Algorithm was not rectified to fit the new ecosystems. Applying ILPG means converting the whole anonymisation program to one UDF program. In another word, Parallel distributed framework will not be able to process ILPG efficiently, instead, data will be transferred and processed in a UDF, which is outside the parallel distributed framework.

So far, two critical concerns in the current anonymisation methods. One more critical issue is regarding the lack of scalability in the anonymisation program. The current programs have restricted the number of Q-IDs. Maximum number of 9 Q-ID attributes can be assigned to each dataset. The increased number of Q-IDs may require an intensive computation cost. Imagine the ILPG needs to calculate the score for each attribute before deciding the specialized one. Hence, more Q-IDs will definitely reduce the speed and performance. We need an anonymisation method that is able to use many Q-ID attributes. The need for increasing the number of Q-IDs and auxiliaries is high. The recent evolution of social media and portable hardware service have urged developers to increase the number of Q-IDs in multi-dimensional data. Adversaries can easily identify a person by searching the internet for some details about the person’s posts and a profile from Facebook, Twitter, Linked-In and others. We need an anonymisation framework that is able to deal with many Q-ID attributes efficiently.

Increasing the number of Q-IDs in multi-dimensional dataset supports security and granularity. The increase demand on data analytics imposes better tools to deal with the authorization level and access control. More demands on data analytics means more users requesting access to data. The current anonymisation methods cannot be considered as granular access control methods. The anonymisation is applied evenly to all users. There is no any gradual access control for multiple users. The advancement of access control techniques exploits gradual and fine-access control to improve the security level. Since data owners may share data with business partners, strategic partners, co-owners, contractors, and public. User’s business background can determine the security level for each individual organization or user. The granularity can be applied on the level of anonymisation. Hence, a user with a high security access is prone to a high level of anonymisation, and vice versa.

## Summary

This chapter introduced the background of big data analytics and the challenges that face the big data analytics security at the present and in the near future. The chapter initially introduced big data framework structure, and the recent tools of managing such a massive size of data. Three layers of big data structure are introduced; infrastructure, computation, and application. The infrastructure refers to the hardware equipment in the data centres. The computation refers to the middleware of file systems on managing files and NoSQL on managing the big database. The application layer presents the parallel distributed frameworks that process queries of data analytics and cluster operations. The scope of this research focuses on data analytics security, hence, data analytics was introduced by manifesting its importance in big data, and the challenges that face analytics. One of the analytics challenges is the processing performance and speed. Data analytics algorithms tend to operate intensive computations probabilities and statistics. Analytics tools, such as MapReduce, still have limitations in accomplishing such large jobs in a real-time manner. This is the major concern in big data analytics. Parallel computing frameworks conclude some tools were specially designed for batch processing, while other frameworks were designed for batch and stream processing. However, streaming tools are unable to operate efficiently in big data analytics. Two frameworks were introduced for each processing type, Pig for batch processing and Spark for stream processing.

The next following sections delved in data analytics and security. The sections reviewed both of differential privacy and *k-anonymity*. Several methods supported the differential privacy such as, Airavat and GUPT. Other methods support *k-anonymity* in traditional data and big data. The major difference between differential privacy and *k-anonymity* is the interactivity for each type. Differential privacy is an interactive form, while *k-anonymity* is a non-interactive form. The interactive form exploits statistical results without revealing data, in contrast to non-interactive form, which permits a partial data-access after anonymising some data values. One major challenge in differential privacy is the user’s queries. Applying some obfuscation to the statistical results requires a full understanding to user’s queries. Some adversaries may intentionally create risky queries, so they can avoid the obfuscation. Researchers spent a considerable time on predicting and finding proper solutions for attacker’s queries.

Two techniques of *k*-anonymisation were explained; generalisation and specialization. The technique of generalisation is opposite to the specialization. The generalisation implements the taxonomy tree to move from the bottom of the tree up to the root of the tree. Therefore, it is known by Bottom-Up Generalization or BUG. The specialization implements the taxonomy tree to move from the top of the tree to the tree bottom. Therefore, it is known by Top-Down Specialization or TDS. Both BUG and TDS algorithms operate efficiently in traditional data. However, big data anonymisation by TDS or BUG is inefficient for several reasons such as: data is split randomly, current anonymisation methods do not cope with the latest technology of parallel distributed operations, and the scalability lack of the anonymisation program.

# - MULTI-DIMENTIONAL SENSITIVITY-BASED ANONYMISATION METHOD

Data analytics and their utilization in big data environments witness a rapid growth in the past few years. Several undesirable side-effects have appeared, in relates to data disclosure and privacy violations risks. This trend imposes finding privacy methods with a scale-up ability to cope with the big data growth. Data anonymisation is one of the pioneer privacy solutions that can minimize such risks. However, the current anonymisation solutions suffer from poor performance and high loss of gained information in the big data environment. This Chapter introduces a novel privacy method named as Multi-Dimensional Sensitivity-Based Anonymisation. The method resolves the performance and anonymisation loss concern and provides a role-based anonymisation control. Various privacy methods were proposed to anonymise data before exposing sensitive information on the cloud. The contemporary anonymisation methods do not take the big data specifications into considerations.

Big Data analytics is where advanced analytic techniques operate on big datasets [83]. Hence, analytics is the main concern in big data, and it may be exploited by data miners to breach privacy [84]. In the past few years, several methods that address the data leakage concerns have been proposed for traditional data [73, 85]. The proposed methods provide remedies for variant types of attacks against data analytics process. Side attack is considered to be one of the most critical attacks [6]. Side attack method was explained in Chapter 2. This attack is prevalent in medical data, where the attacker owns partial information about the patient. The attacker aims to find the hidden sensitive information by logically linking between his/her data and the targeted data [86].

The previous Chapter has addressed both traditional and big data possible attacks and solutions. However, new methods did not consider big data specifications and behaviour. The proposed anonymisation methods are inadequate for big data operation. This is correct even for the recently proposed anonymisation methods. Recent methods, such as Two-Phase Multi-Dimensional Top down Specialization method (Two-Phase MDTDS) [87] , MapReduce TDS (MRTDS), or MapReduce BUG (MRBUG), do not mimic parallel processes and operations [25]. The mentioned methods can operate efficiently in traditional data. However, big data specifications and operations concepts are different. Big data operates in a parallel distributed environment, where performance and scalability are a major concern.

There is a need for an anonymisation method that is able to operate in parallel. This imposes a need for changing the core structure of anonymisation technique. Before proposing any anonymisation method for big data, some specifications should be considered. Developers need to distinguish the disparity between big data and traditional data. . With big data, anonymisation process should be able to reduce the computation costs, prevent high information loss and increase security. Also, the anonymisation process should be provided with a granular access control method. Hence, any big data anonymisation developer should pay attention to the following specifications: equivalency increase, information gain, parallel algorithm, and gradual access. These specifications will be discussed in the next sections.

## Requirements for Big Data Anonymisation Method

### Equivalency Increase

The equivalency increase is a general specification that must be considered on proposing any *k-anonymity* method for big data. This can be defined as;

**Lemma:** *In data records, the percentage of equivalent records proportioned extrusive with the increasing number of records. The rising number of records can help the least common attributes to gain the equivalency.*

This is true for most attributes. Few attributes are excluded, as a reason for their solitary nature like; emails, usernames, phone or fax numbers, and primary keys. To prove Lemma mathematically; let us denote the number of occurrences for any Q-ID value by *n*. Therefore, the Q-ID probability is calculated as; P(*qid*)=1/*n*. If the number of Q-IDs is *m*, then the number of occurrences of any Q-ID record is calculated by the factor of the Q-IDs;

(3.1)

Where P[QID] is the probability of each Q-ID.

For each record of Q-IDs, the number of occurrences is given by. If we assume that each combination of values appears only once; then we need at least n records to gain one-time occurrence. Also, we need *k n* records to gain the *k-anonymity* for each combination of values. Referring to *k-anonymity*, the equivalency q is defined as the total number of equivalent records, where q ≥ *k* for each occurrence. For instance, if *k* =5, then each distinguished record must appear five times in N before gaining the *k-anonymity*. In the real data, the appearance number of Q-IDs combinatory is usually less than n. Let us call the appearance number of actual combinations is n̄. Based on our assumption of one time appearance for each combination; we can calculate the minimum value of N as:

‎3.1

Where n̄ denotes the number of actual combinations that appear in N, n̄ ⊂ n.

Equation 3.1 assumes that each record has an equal number of appearances to the other records, which concludes that q=*Nmin*. However, in the real data, this is not a typical case. Thus, some records appear less frequently than the others, which makes some records reach the equivalency, while others fail. However, Equation 3.1 describes only one scenario. Nevertheless, any situation should consider the variable n̄. The probability value of variable n̄ remains between stability and increase, and it never decreases. In reality, the value of n̄ usually increase, while the stable scenario is less common. Besides, the equivalency q is proportioned extrusive with N, and can be described as q ≤ N, so q α N. This can be presented by the increase percentage of equivalency Q = q / N.

The positive relationship q ≤ N can be proven experimentally. Three various experiments were conducted by using the adult database from the UCI Machine Learning Repository [88]. The database describes the Age of adults, their Occupation, Marital status, Education, Social status, Position, Sex, Hours worked per week, Race, County, Native country, and salary. Four Q-ID attributes are assigned as the following: Age, Education, Salary, and Sex. The experiments are conducted using MatLab simulator [89], by choosing three groups of N records small, medium and large.

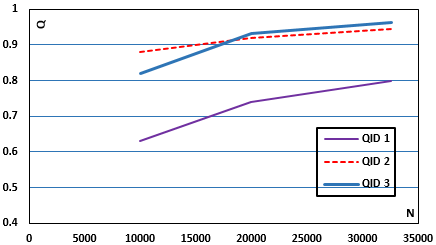
During the experiments it is assumed that *k* = 10, and the Q-IDs probabilities are calculated as; P[age] = P [1-100] =0.01, P[education] = P [Y5-6, Y7-8, Y9, Y10, Y11, Y12, HS-grade, Some-college] = 0.125, P[sex] = P [Male, Female] = 0.5, and P[S] = P [<=50K , >50K] = 0.5. Hence, the maximum number of combinations is calculated as.

In the first experiment; the total number of records was N =10,000 records. The number of the actual combinations appearance in 10,000 records was n̄ = 1741, which presented around 50% of the probable appearances. The number of equivalent records was q=6272, which presented around Q=60% of the total number of records. In the second experiment, the number of records was increased up to N= 20,000. The number of actual appearing combinations was n̄ = 2196, which presented around 69% of the probable appearances. The number of equivalent records was q=14828, which presented around Q=75% of the total number of records. In the third experiment; the number of records was further increased up to N=32,561. The number of actual appearing combinations was n̄ = 2498, which presented around 78% of the probable appearances. The number of equivalent records was q=26846, which presented around Q=82% of the total number of records.

Similar steps were applied on three groups of {G(QID)1, G(QID)2, G(QID)3} randomly picked Q-ID’s in the same dataset. The three groups are mentioned in Table 3.1. The Q value was calculated for each group. All groups showed an increase in both of the equivalency percentage Q, as shown in Figure 3.1, and the actual appearing combination n̄, as shown in Figure 3.2.

Table ‎3.1- Four Q-ID groups chosen randomly from Adult dataset

|  |  |  |
| --- | --- | --- |
| **G(QID)1** | **G(QID)2** | **G(QID)3** |
| Age | Marital status | Position |
| Education | Education | County |
| Sex | Social status | Country |
| Salary | Race |  |



*Figure ‎3.1- Proportionality of equivalent records with increased record numbers*

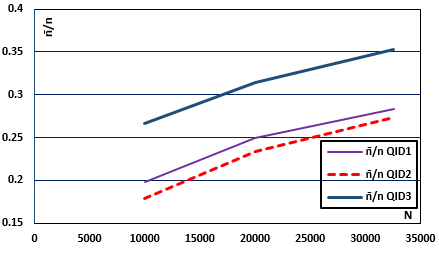


Figure ‎3.2- The effect of increasing the number of actual combinations with increased number of records

Both diagrams prove that data growth supports data equivalency in big datasets. For this reason, we need to find a proper algorithm that is able to process a large data size without a need for splitting data into small chunks with a large data processing capacity.

### Information Gain and Security

Current anonymisation methods are mostly adaptations of approaches that were designed for traditional data [90]. The traditional data concludes a limited number of records, therefore, information gain is an essential matter. However, big data equivalency increases extrusive with the data size increase, which concludes a larger group of equivalent records. Thus, the information gain concern may not be the prime factor in anonymisation process. Logically, more equivalent data cause less anonymised records. Also, all indications point out security concerns in big data. The increased number of users’ access and the massive quantity of personal information in datasets may increase the re-identification probability. Hence, the trade-off between security and information gain may result in favour of security. Therefore, anonymisation algorithms should pay more attention to security concerns. Algorithms with several times iterations, to find out the best Q-ID anonymisation, the best *k* value, or the best cut and interval are inadequate in big data. The multiple iteration and scanning is an expensive computation cost. Moreover, the high level of accuracy and the small value of information gain increase does not really affect the statistical and analytical results.

One of the major distinguishing features in big data is the multi-dimensionality [91]. This leads to a large number of Quasi Identifiers. The previously proposed methods suggested limited number of Q-IDs, which may reach up to seven or eight. However, when data is multi-dimensional, then more expected Q-IDs will join the data records. The large number of Q-IDs leads to a higher security threat. Re-identification becomes easier with the increased number of auxiliaries and data identifiers. More personal information revealed will definitely facilitate the side link attack or the background knowledge attack. The attack possibility becomes even higher with the new technology evolution of social media, smart phones and cloud services. Users are now able to find out most personal details over the internet. Hence, more security vigilance is essential.

The above facts do not mean a total ignorance for the information gained by the anonymisation method, but reducing the processing of finding the optimal anonymisation values instead. The optimal values usually lead to the lowest possible loss of information after anonymisation. In big data, applying such complicated processes may not affect the final results of statistical output. The small statistical values can be even ignored, since the statistical results follow the principle of estimation prospects. This gives data miners a flexibility of approximating and rounding some numbers to few decimal places [28]. Therefore, pre-calculating the *k* value, and pre-determining the attributes needed to be anonymised is an advantage. Generally, this non-accuracy will not dramatically affect the data analytics results.

Moreover, big data is beneficial when it is public, this means many organizations from different fields need to access this data for multiple purposes [92]. They all analyse, mine, and output statistical results. This fact emphasizes the security manifestation side in big data. Multiple users from various organizations may need to access data with different levels of access. Users cannot be given similar security level, therefore, granularity is required as a part of security procedures. This will be further explained later.

### Parallel Algorithm

A parallel distributed environment handles big data. The multi-task processes should be considered in any anonymisation method used for big data. This can be implemented by splitting tasks into sub-tasks, and distributing them among multi-computers to cope with the massive data volume [93]. Parallel programming model is provided by the processing framework such as; MapReduce or Spark. The framework provides parallel operations for reading/writing data from/to disks, storing data in memory, and processing data in parallel. These three procedures can be completed by the big data frameworks. Parallel programming model is classified into two areas; process interaction and problem decomposition. The process interaction is related to managing shared memories to accommodate the processed data, passing messages between nodes, and dividing large tasks into smaller subtasks. The problem decomposition is related to the skeleton of the algorithm, and the programming paradigm and sequence. This concludes that the efficient parallel framework, and the proper algorithm and programming sequence should lead to a successful and well performed task [94].

Many obstacles face parallel programming models, such as; network speed and massage passing between nodes. This causes a bottleneck of sequential operations in shuffling phases. Also, finding a general valuable programming model is difficult. Any parallel model is judged on its generality, since some models may perform efficiently in specific data types and environments [95]. For instance, Flink and Storm frameworks perform well in multimedia and data stream, while Hadoop Distributed File System performs well in large data files. Parallel programing adopts the implicit and explicit parallelism concepts, whereas implicit parallelization converts the program codes from sequential into parallel automatically. The explicit programming is difficult and requires high programming skills. Programmers need to code all short messages and shared memories across the cluster nodes. For this reason, current parallel models for big data are partially implicit [96].

In big data, the processing framework, such as MapReduce, handles the majority of the implicit parallelization. However, some parts of the parallelism rely on the application programmer, who should be aware of the framework core structure. The programmer needs to know the optimal code that may result the best performance. This can be implemented by several trials of running the program, to arrange the commands of filtering, grouping, replacing and others in the best sequence. In big data anonymisation, the programming commands control the dataflow operands without changing the program’s state. The programmer no longer specifies the detailed sequence of instructions in execution orders, but rather the general operations to be applied to the dataflow. Eventually, the programmer has a small window of managing the operations sequence, hence, the anonymisation framework should consider this level of programming.

### Gradual Access

Any proposed big data framework should consider the large number of demands on analytics. The more data growth, will lead to a higher number of analyser’s requests. The large data size is mostly generated by a large number of participants; hence, they contain a great percentage of personal information. As a result, analysing this massive data will enhance the future plans and strategies in marketing, development, and decision making [97]. Because big data nature is public, and various analysers may wish to participate in data analytics, then there is a need for considering the security term as the first priority. One of the main security components is the fine-grained access control for users. This implies a gradual access as per user’s right and privileges.

The gradual access should be provided through the anonymisation control, by increasing or decreasing the anonymity level. Users with high privileges should be able to gain higher information levels. Therefore, the statistical analytics results should be close similar to the original data analytics results. In contrast, users with low privileges should gain lower information levels. This data masking increase or decrease keeps data in the hands of the authorized users and organizations. The access control should be able to hide some data from certain users. Hence, the access control can view or hide data only. The view is masked with levels of data anonymisation. The granular access should not have a large impact on analytics performance.

## Multi-Dimensional Sensitivity-Based Anonymisation Method Concept

Multi-Dimensional Sensitivity-Based Anonymisation (MDSBA) method is developed to fulfil the previous four specifications. These are equivalency increase, information gain, parallelism, and user’s access disparity. MDSBA adapts a multi-dimension technique for performing a high level of computation in big data. The MDSBA method mandates to define the privacy method and masking pattern for each access level. The novel anonymisation method applies the Bottom-Up Generalization (BUG) in *k-anonymity* that can cope with the big data frameworks. The method does not only parallelize data for big data frameworks, but also reduces the overhead computation of data iteration. This is accomplished by providing pre-calculated *k-anonymity* parameters and pre-determined attributes for anonymisation. The MDSBA also supports the anonymisation-based access control. This imposes a gradual anonymisation based on user’s access level. MDSBA mimics role-base access control by providing a granular security access for multi-user levels.

MDSBA consists of four main concepts, these are the definition of datasets, the probability concepts, the grouping, and the sensitivity value. Following this introductory, let us delve in MDSBA core concepts.

### Quasi Identifiers and Classes

The MDSBA general definition is similar to the previously proposed *k-anonymity* methods. Data owners choose some private attributes as Quasi Identifiers (Q-ID). The chosen Q-IDs contain personal attributes that may facilitate adversary’s tasks of re-identifying some records. More importantly, they may unveil some sensitive attributes, known by classes. For this reason, some of the chosen Q-IDs will be anonymised, if they do not fulfil the *k-anonymity* rule. This imposes three different types of data attributes; Q-ID attribute, sensitive attribute (class), and ordinary attribute. This can be defined as a table T consists of *m* set of attributes, where each attribute can be either Q-ID, class C, or an ordinary attribute *attr*. Hence, T is a combination of T= {QID, C, *attr*}.

The chosen Q-IDs and C are pre-determined by data owners. There is no clear instructions on picking up these identifiers. MDSBA provides the best practice advices for determining Q-ID’s. This will be further explained in chapter six. Data owners setup the initial values of their own, including the Q-ID attributes decision. Data owners are the only users who are permitted to access the original copy of data exclusively. The anonymisation level increases gradually with the ownership increase.

### Probability Concept and Anonymisation Masking

MDSBA adopts Q-ID probability concept on aggregating the Q-IDs and on masking process. The anonymisation is conducted by applying one of the three masking tools, taxonomy tree, interval, or suppression. The probability is pre-calculated for each Q-ID related to the possible values that may appear in all data records. For instance, the attribute probability of a person’s age is P(age)=1/100, if we suppose that the human age range is between 1 to 100. The Q-ID probability should be pre-calculated to accelerate the grouping in the anonymisation process. The number of Q-ID values indicates how often the arbitrary appearance of each value. If the values are unique or non-repetitive, then the attribute cannot be an elected as a Q–ID attribute. Thus, the values must be finite and repetitive. Two concepts, in MDSBA, adopt probability approaches, these are:

* **Probability in aggregating Q-IDs:**

As mentioned before, the Q-ID probability is essential in grouping records. The grouping, in MDSBA, is conducted on a gradual basis. This allows grouping for all Q-IDs first, and then another grouping for all but one, and then for all but two and so on. In general, the lower Q-ID probability values may appear less frequently in data attributes. This imposes that data records with high Q-ID probabilities may return less data equivalency, and as a result higher anonymisation output. For this reason, MDSBA meant to reduce the anonymisation output by grouping the Q-ID attributes with the highest probabilities. MDSBA, divides the grouping task into multiple tasks, by grouping all Q-ID attributes in the first stage, so this can filter out the fully-equivalent records. In the second stage, the lowest probability is excluded from the second grouping process. In the third stage, the lowest two Q-ID attributes are excluded from the third grouping process. The final stage aggregates only one Q-ID attribute, which is the one with the highest probability.

* **Probability in Masking:**

The probability is also used in masking Q-ID attributes. This probability is derived from the taxonomy tree concept. The taxonomy tree T is propagated from the parent node w to a number of leaf nodes ν, so each parent node’s probability is. Figure 3.3 illustrates probabilities for each parent node in the education tree. For instance, the parent (Primary) has three children nodes, which concludes a probability of P= 1/3 ≈ 0.33. The probability concept can be applied on taxonomy and interval masking. The two major anonymisation tools are presented by taxonomy tree and intervals. This depends on the data type, since some data can be generalised by taxonomy trees, such as (Education). In contrast, other data types are numerical, and can best fit interval masking. The intervals can be presented by probability values. If a number n was presented in an interval of a minimum value *Vmin*, and a maximum value *Vmax*, then the probability of obtaining that number within the interval range is. For example, for an interval of [15 – 25[, the probability is (P=1/10=0.1). This probability concept supports the fine-grained access for multiple users.

#### Interval and Taxonomy Tree Masking

The probability, in MDSBA, is the core concept of giving a gradual level of information gained from data. Users with higher privileges are given more accurate data due to their high trust level. The higher trust they have will lead to a higher information gain. The trust level is determined by several factors that will be discussed later. To illustrate the granularity in information gain, let us consider Figure 3.3 again. The higher trusted users may receive data masked with level one taxonomy tree. This imposes more accurate information about people’s education level. For instance, if a user was given an access level with a maximum probability of P(user)=0.17, then data sample appear in Table 3.2-A will be anonymised as per EDU taxonomy tree. The anonymisation results, as illustrated in Figure 3.2-B, show that one value was masked by level 1, while the rest of the values were masked by level 2. This is because the probability of the secondary node in the tree is 0.17, and the certificate probability is 0.067 < 0.17. In another example, if a user’s maximum probability P(user)=0.5, then the given masking will reveal more information, as shown in Table 3.2-C.

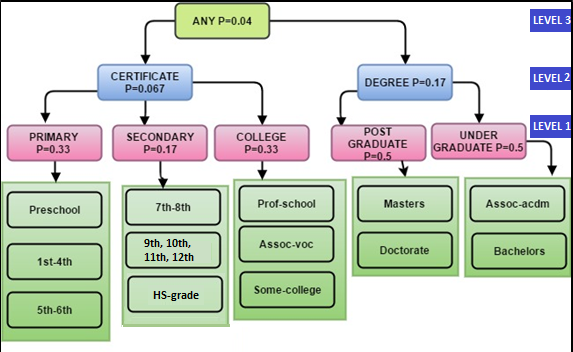
****

Figure ‎3.3- Taxonomy tree for EDU in Adult data.

Table ‎3.2-A. Original Data Table 3.2-B. Masking with P ≤0.13 Table 3.2-C. Masking with P ≤0.5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EDU** |  | **EDU** |  | **EDU** |
| Master | Degree |  | Post Graduate |
| Bachelor | Degree |  | Under Graduate |
| 9th | Secondary |  | Secondary |
| Some-college | Certificate |  | College |
| 5th-6th | Certificate |  | Primary |

In the previous examples, users are given probability values to control their access level. However, we need a mechanism to assign this probability value. In order to control the granularity, let us define an instant of *k-anonymity* value that is given for each user. Suppose that the *k-anonymity* value is *k1*, while another user was given a value of *k2,* where *k2 < k1*. In this case the smaller values of *k1* impose less anonymisation impact. Therefore, the user with *k2* value will gain more information. The probability value**,** given to each user, can be better controlled by given multiple values of *k-anonymity* as per user’s access level. Thus, users with higher values of *k* may gain less information than users with lower values of *k*. This enhances the probability concept and provides better tools to control the access.

Referring the *k-anonymity* value and the granularity, let us define an instant of *k*, denoted by, and can be named as the ownership level. It is a parameter given to each user on accessing data for analytics. This number indicates the minimum number of equivalent Q-ID records to avoid anonymisation. Larger value of implies a higher level of anonymisation. The probability value for each user can be given a name of Sensitivity level, and is denoted by ψ. The sensitivity level ψ is calculated by giving the ownership level for users. Data owners determine the Q-ID attributes, and the value of *k*, then, the level of sensitivity is determined by MDSBA equations as per given for each user. The Sensitivity Level of the class attribute C is denoted by ψ, and the ownership level of a user is indicated by  = *i*-*k*, where *i*= {*k*+2,*k+3,… , k+k*}, and 2≤ ≤ *k*.

#### Suppression Masking

Data can be processed by applying three different types of masking, taxonomy tree, interval and suppression. The suppression imposes a full or partial data hiding, so zero or low information are gained for the suppressed data. MDSBA implements suppression in some cases, such as non-equivalent records after completing all grouping stages. This is usually applied during the final grouping stage, when grouping cannot find any semi-equivalent records. Moreover, suppression can be a proper solution for some data types. For instance, data of postcode, first name, last name, and contact numbers. These data better anonymised by applying special characters to replace the actual value. The common used character is the star (\*). Also, the suppression can be applied by using the word (any).

The probability can be applied with suppressing masks. For instance, if a person’s maximum probability is 0.04, then the name (Mark) can be masked by one letter and displayed as (Mar\*). The omitted letter’s probability P(letter)=1/26=0.04. Similar concept is applied on postcodes, for instance, a postcode of (4514) manifests a value of 0.1 for each suppressed number. Hence, the previous postcode can be given by (45\*\*) for the same user. This concept is quite similar to intervals and taxonomy trees. However, some data types may accept either suppression, interval, or taxonomy. Therefore, data owner needs to determine the masking tool used in each data attribute. For example, the previous postcode can be anonymised by an interval of [4510 – 4535[. If compared between both masking results, then the interval may produce more accurate anonymisation output. In the previous examples, the suppression process has omitted two numerical values from the postcode. The probability of the omitted numbers = 0.01, while the user is given a probability up to 0.04. However, it is not possible to omit one number of the postcode, because the probability of omitting one number is P=0.1, while the maximum given probability is P=0.04. On the other hand, the interval masking can accurately assign the exact probability. In the previous example, the user’s probability is 0.04, which is presented as an interval of 1/0.04 = 25. The postcode can start from 4510 with an interval of 25, so the maximum interval value is 4535, so the interval can be [4510 – 4535[. In case if the interval or taxonomy are more accurate, then it is better ignoring the suppression masking, or keeping suppression as a last resort.

In some cases, it is possible that user’s given probability is much lower than the probability of the taxonomy tree root. In such cases, the probability concept is ignored, and data is fully suppressed. For instance, if the maximum probability P(user)=0.01was given to a user. Referring to the previous EDU taxonomy tree, in Figure 3.3, the root probability of P(Any)=0.04. Based on the probability concept, the EDU data cannot be anonymised by the value (Any), because P(user) < P(Any). However, (Any) is the highest level in the tree, and there is no any further generalisation can be applied. Therefore, the masking algorithm generalises any EDU value with (Any), if the user’s probability P(user) ≤ 0.04.

### Grouping Data Vertically and Horizontally

In order to parallelize massive size of data, and to support the anonymisation granularity, MDSBA logically splits the Quasi Identifiers (Q-ID) vertically and horizontally. This technique supports several aspects in big data. In section 3.1, four different requirements were identified for any anonymisation method. The grouping concept mitigates the anonymisation costs, and increases the security level. It supports the anonymisation requirements of equivalency, security, parallelization, and gradually. Vertical grouping divides the Q-IDs into small groups, so they can be an advantage for big data anonymisation. When grouping data horizontally, more attributes can be chosen to be Q-IDs, which enhances the security level of the anonymised data. This technique splits large data into smaller data size to support the parallelism. Each group of data is sent to a separate node, and processed in parallel.Vertical grouping enhances the access control method. Since users will be give the needed Q-ID groups only. The non-needed groups will be hidden, so users will be given the right amount of data only. Eventually, vertical grouping reduces he computation costs, so the computation process is conducted individually for each Q-ID.

Horizontal grouping enhances the information gained by splitting data logically instead of random split. This is further explained in chapter 5. Moreover, this grouping mimics the MapReduce framework by staging data from map to reduce. In horizontal grouping, data is grouped several times, and in each time, data becomes smaller. In the horizontal grouping, algorithms do not need to load a massive size of data with multiple iteration times, instead, data is dramatically reduced on each iteration time. This grouping method enhances the information gain, the equivalency increase, and the granularity. Let us further study each type of vertical and horizontal grouping.

#### Grouping Data Vertically

MDSBA splits Q-IDs in to small groups of two to four Q-IDs. The groups are chosen based on the business roles. Personal information and related auxiliaries can be categorized based on user’s interests. In MDSBA, the aim is including all or most personal attributes in Q-IDs. This increases the total number of Q-IDs, so there should be a way of dealing with this large number of Q-IDs. Referring to user’s various needs and to the increased number of Q-IDs, it is possible that each Q-ID group can be mapped to a business role. For example, Human Resource Administrator may focus on the patient’s address, age, and salary, while Radiologist may care about the cancer’s type, status, and size. The mapping between Q-ID groups and roles will be discussed in chapter 5.

We may describe the vertical definition by denoting n as a number of records, and m as a total number of attributes. The data owner defines the number of quasi-identifiers as *Q* in *m* attributes, so *Q* ≤ *m*. Each two to four Q-IDs are aggregated in a group G(QID), so the number of created groups, denoted by ϒ, is related to the total number of Q, and can be presented by , where . Each non-overlapped G(QID) group consists of Q-IDs and one class, which is usually mapped to one or more business roles R. Let us also denote U as a user, so the role-based anonymisation control is presented as; {G 🡪 R}(many to many relationship), and {R 🡪 U}(many to many relationship). The vertical grouping principle is described by the following definition;

**Definition 3.1**: *a table T contains a Q number of Q-IDs, and a C number of classes. Q-IDs are grouped vertically by dividing Q and C attributes into* ϒ *groups. Each G(QID)* *group contains one class attribute and some two to four Q-ID attributes. In other words, 2 ≤ G(QID) ≤4, and C= 1.*

The vertical grouping divides Q attributes into small groups of two to four Q-IDs, with one class attribute. Each Q-ID group, G(QID), is mapped to a business role, and given a fixed value of *k*. In such a way, users are given authorization rights to access some Q-ID groups as per their given business role. Let us study the following example to illustrate the access control structure. If five users={user1, user2,…,user5} have requested to access the following two Q-ID groups; G(QID)1={admission\_date, cancer\_found(yes/No), diagnosis(class)}, and G(QID)2={age, job, suburb, salary(class)}. G(QID)1 is mapped to Doctor, and given a value of *k*=20, and G(QID)2 is mapped to Finance Manager and given a value of *k*=30. Suppose that users are given the following roles {user1(Doctor1), user2(Doctor2), user3(F. Manager1), user4 (F.Manager2), user5(Doctor+F.Manager). Each one of these users will be given a value of to represent the ownership level. The values are shown in Figure 3.4. This figure illustrates the core base of access control management in MDSBA. The access granularity is presented by giving the G(QID) groups fixed *k* values, and giving users variable values of . The *k* value determines the maximum optimal value for each group G(QID). Chapter 6 illustrates the method of finding the optimal k value. Each user is given a value of , which determines the user’s access level. Users with higher access privileges obtain low values of , and vice versa. In Figure 3.4, Doctor1 gains more information than Doctor2 on accessing G(QID)1, while both doctors are not permitted to access G(QID)2. Similarly, Finance Manager 2 gains more information than Finance Manager 1 on accessing G(QID)2, while both managers are not permitted to access G(QID)1. The figure also shows the possibility of assigning more than one business role to a specific user. In Figure 3.4 user 5 has the permission to access both groups, since he/she was assigned to both business roles. User 5 will gain high information from G(QID)1 and low information from G(QID)2.

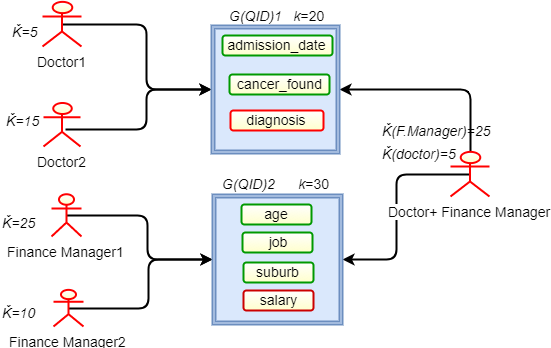


Figure ‎3.4- Access Control structure in MDSBA

#### Grouping Data Horizontally

It is evident that records equivalency, in most datasets, increase in parallel with data growth [98]. Therefore, splitting data nominally may increase the data equivalency. Random data split ignores the distribution of equivalent records amongst tables. For instance, Adult data that contains the following attributes: patient’s age, sex, and education attributes may appear similar in the first one thousand records, and then related records may appear again at the end of the table. Hence, the random appearance of similar records may reduce the number of equivalency on splitting data into small random chunks, which results in an increase of data masking.

The following four definitions describe the horizontal split of data by implementing the grouping method of equivalency in MDSBA. It was proven earlier that splitting data nominally will increase the equivalency ratio. Suppose a data set D, with a total number of *m* attributes, and *n* records. The data owner defines the number of quasi-identifiers as *Q* in *m* attributes, so *Q* ≤ *m*. Let a number of *q* attributes have equivalent records *k*, where *q* attributes are part of *Q*, so *q* ≤ *Q* ≤ *m*. Hence, the wholly or partly similar records are defined as follows:

**Definition 3.2:** *All D records are split based on the class C values. Each set of records that contains similar class value is aggregated in a Gi group, where i denotes the number of values appear in the class C. Every Gi group is further processed individually.*

**Definition 3.3:** *The fully-equivalent group (SG) contains some k equivalent records, in some Q attributes. For this group, there is no any anonymisation process applied.*

**Definition 3.4**: *The Semi-equivalent group (SSG) contains some k equivalent records, in some q attributes, where 2≤ q ≤Q-1. The highest Q-IDs probability is usually chosen for q attributes equivalency. The anonymisation is applied on the rest of non-equivalent Q-IDs.*

**Definition 3.5**: *The non-equivalent group (NG) contains a number k of equivalent records, in some q attributes, where the number of q=1.The highest Q-ID probability is usually chosen for q attribute. The anonymisation is applied to the rest of non-equivalent Q-IDs.*

As explained in Definitions 3.4 and 3.5, the SSG and NG anonymisation are applied on the lowest Q-ID probabilities. For instance, the Q-IDs probabilities, in Adult data, are; P[Age]= 0.01, P[Sex]= 0.5, P[Edu]= 0.08. Based on these values, the SSG anonymisation will be applied on the Q-ID with the lowest probability, which is [Age]. So the semi-equivalency is measured by grouping Sex, and Edu attributes, while Age attribute is anonymised as per interval. The NG anonymisation will be applied to all Q-IDs except the one with the highest probability. In our example, the anonymisation will be applied to [Edu] and [Age], while records are grouped as per [Sex] equivalency.

MDSBA framework splits data recursively to create three dependent stages of MapReduce processes. Stage one aims to produce Gi groups for the whole dataset with one Pig command only, which filters each data record based on its class attribute, as described in Definition 3.2. The number of the produced Gi groups relies on the number of sensitive class values. For instance, Seer Cancer data contains four values of sensitive attribute (Cancer); these are {no positive histology, Positive histology, Positive microscopic confirm, and Positive laboratory test}. The four values create four groups of Gi ={G0,G1,G2,G3}. Each group is stored in a separate HDFS location. Stage two reprocesses G groups in parallel to categorize data records between wholly or partly equivalent, or between SG and SSG. According to definitions 3.3 and 3.4, the SG groups denote the equivalent Q-ID group, while the SSG groups denote Semi-equivalent group. The SG groups are equivalent, so anonymisation is skipped, and data is stored in an output directory. The third stage’s input is derived from the second stage’s output, so SSG groups are stored in an input directory waiting for stage three. The process starts by reading SSG groups and abstracts the non-equivalent groups, denoted by NG. Both of NG and SSG are anonymised by using User Defined Function Java program.

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

### Mathematical Equations to Calculate the Sensitivity Level

As explained before, data is split into several groups or domains vertically and horizontally. In each Q-ID group, a different anonymisation is applied depending on the user’s access level. The anonymisation process is managed by the value of sensitivity level ψ, which increases or decreases the information gained from the data. User with a higher value of ψ gains more information, and vice versa. The sensitivity level is determined by two major factors: the sensitivity factor ω and the aging factor τ. In datasets, the value of represents the *k-anonymity,* and represents the ownership level. A large value of ownership level implies a weak ownership relation, so the weakest ownership relation is when. The integer values of and are such that. In other words, low values of correspond to reduced anonymity as a result of higher ownership relations. The sensitivity factor ω is calculated based on its maximum and the minimum probability values. The maximum probability value of ω is defined as:

‎3.2

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

‎3.3

Based on Equations 3.2 and 3.3, the value of ω can be found linearly between ωmin and ωmax, as shown in Equation 3.4:

‎3.4

Equation 3.4 presents a linear relation between ω andvalues. The equation lowers the value of ω when the user’s value is high, so that the data anonymity is higher. Users with less privileges are given higher, while the value of *k* is constant. It is also proposed that the lowest value of should be two to avoid unique re-identification. The anonymisation is only applied on external organization’s access. External organizations are permitted to access an anonymised copy of the original data, while accessing the original data is exclusive to the data owners.

Equation 3.5 collates both terms of ω and τ to compute the sensitivity level ψ. The sensitivity level directly manages user’s access privileges. Higher sensitivity levels lead to upper access ranks, where less data masking and concealments are applied. Also, the sensitive nature of an object can be made lower as its related data set ages. The aging factor τ affects the sensitivity reversely, with considering the negative values of aging factor. Based on their owners’ decisions, old objects can be deemed to be less sensitive, and increase ψ as per age. In essence, two factors determine the sensitivity level of the data, the sensitivity factor ω, and the aging factor τ, as described Equation 3.5.

‎3.5

Equation 3.5 is used to calculate the sensitivity level ψ, by establishing the sensitivity level of an object for a given user access level. The masking process tends to find any number close or smaller than the sensitivity value. For instance, if ψ =0.5, then any value between 0-0.5 is acceptable. However, the values closer to ψ improve the information usefulness and the granularity precision. For instance, the age interval of [10 – 20[is derived from ψ =0.1, while the age interval of [10 – 12[ is derived from ψ =0.5. We may notice the accuracy difference between both values of ψ, therefore, assigning the closer value to ψ is essential. The aging factor creates a perturbation to the sensitivity value. This factor becomes more significant as the data age becomes older than a particular age that we refer to as its obsolescence value. Equations 3.3 and 3.4 show that the lower sensitivity level requires a higher value of, which corresponds to lower information gain. In other words, the lower sensitivity levels, correspond to higher anonymisation and masking levels of information.

#### Sensitivity Level and Time Factor

The aging factor τ depends on four different parameters, the object obsolescence value Ø, the aging participation percentage ρ, the object age y, and the sensitivity factor ω. The Ø value is defined as the critical age before the object sensitivity starts to degrade. It can be measured by units of hours, days, weeks, months or years, which totally depends on objects obsolescence speed. However, Ø cannot be given a value less than 2, so the value of one year, for example, can be replaced by 12 months instead. The aging participation percentage ρ is an approximation percentage chosen by data owners. It measures the aging factor participation in data objects. The aging factor τ can remain constant or decrease linearly. It remains unchanged when the age of the object y is less than its obsolescence value, hence it remains constant if y < Ø. On the other hand, if the object age is greater than or equal to its obsolescence value, that is y≥ Ø, then sensitivity level increases logarithmically, which in turn decreases anonymisation level. These two cases manage the aging factor τ, as described by Equation 3.6.

‎3.6

Data owners may set ρ to 0% if their data objects are not affected by time and age. Also, the maximum value of ρ cannot exceed the 90% to avoid a nil value for ψ. It can be noted from Equation 3.6 that τ is always negative. In other words, the sensitivity level ψ increases with the age of the information and the passage of time. The newly created data objects result in a lower sensitivity level compared to the old data. In other words, if *y* <Ø then ψ < ω when the aging factor is incorporated. Equation 3.6 is derived based on our proposed ideas for incorporating the aging factor and sensitivity analysis for improving the anonymisation process. It relies on a linear condition -ρ × ω and a semi-log component (-ρ(y-2) × ω × logy). The linear part produces a constant value of aging factor when the information has not reached its obsolescence value. The semi-log portion is derived from the plotted graph, shown in Figure 3.5. The figure illustrates the concept that incorporates the age of data objects. Older data is considered to have a logistic degradation with age and passage of time. The degradation starts swiftly before plateauing at the sensitivity factor value ω. To illustrate the time factor impact, let us consider the following example: a social media data has an aging participation value set at 0.9 with obsolescence value of 15 days and a sensitivity factor of 0.4 (ρ=0.9, ω=0.4, and Ø=15). Plotting its sensitivity diagram can be initiated by assuming a constant value of sensitivity factor multiplied by the aging participation percentage to find out the aging factor. Therefore, τ=-ρ × ω= -0.36, and ψ=0.04. Next, a logistic graph can describe the sensitivity level direct proportions the variable y increase. The sensitivity level increases dramatically between days 15 and 30. This sharp increase describes the object degradation importance, which will intentionally reduce the obscurity level on data anonymisation.

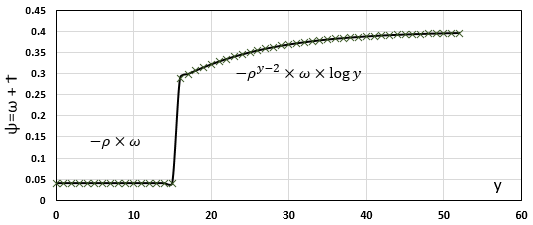


Figure ‎3.5- Plotted graph to derive Equation 3.6 for aging factor

The following example illustrates the calculation steps of sensitivity value ψ. Consider an object with three Q-ID attributes, say student IQ test results, similar to those shown in Table 3.3. The data owner intends to anonymise the data with *k*=20, the obsolescence value Ø is set to 10, the aging participation ρ is 70%, and the age of the object y is13 years. Now, suppose a user was given an ownership value =10. Based on the given attributes, the sensitivity level ψ can be calculate using Equations 3.3 and 3.4. To find out the sensitivity factor ω, it can be noted that the values of ωmax=max (0.01, 0.005, 0.125) is 0.125 and ωmin=0.01 × 0.005 × 0.125 is 6.25 × 10-6. The value of ω as per Equation 3.3 is ω≈ 0.063. Based on Equation 3.6, the aging factor τ=-(0.7)11 × 0.063 × log (13) can be seen to be -0.00138. Therefore, the sensitivity level ψ = 0.063 – 0.00138 is found to be 0.062.

Table ‎3.3- Three Q-ID attributes example

|  |  |  |
| --- | --- | --- |
| **Q-id(attribute)** | **Q-id type** | **Probability** |
| Q-id0(IQ\_value) | Interval IQ\_value=[50-150] | P(Q-ID0)=1/(150-50)=0.01 |
| Q-id1  (Student\_country) | Taxonomy tree Student\_Country\_Level1 = {German, French, Chinese, Kenyan, American…}  Student\_Ancestry-Level2={Caucasian, Asian, Middle Eastern, African, Red Indian…}  Q-id\_level-3= {human} | P(Q-ID1-L2)=1/150=0.007  P(Q-ID1-L3)=1/200=0.005 |
| Q-id2 (Student\_Grade) | Suppression Student\_Grade={A+, A, A-, B+, B, B-, C+, C, C-, D+, D, D-, F}. | P(Q-ID2)=1/13=0.077 |

#### Anonymisation Operations

The anonymisation process can be divided into two main operations, the first operation is completed before the anonymisation process starts, which is finding the sensitivity level ψ. This operation is completed in the FS side. The second operation is masking Q-ID values by either taxonomy tree, suppression, or discretization. The second operation relies on the first operation’s result. Any user requesting data access is subjected to an authorization level evaluation. The evaluation imposes a mathematical calculation for the sensitivity level ψ. The value of ψ is then used to calculate the degree of masking, and the number of equivalent records during the grouping process. The algorithm of anonymisation process is shown in Figure 3.6. It provides a brief description about the anonymisation process. The algorithm shows the parameters that the program needs to retrieve for the entire operations. The program needs to read the users and ψ, in order to compete the grouping and masking operations. Also, the program needs to know other parameters, such as the masking type, the Q-ID probability, the Q-ID names, the class attributes and others. These parameters are pre-prepared and stored in an XML file for each dataset. Hence, two major files are linked with the anonymisation operations; user file and data information file.

The masking pattern is decided by the data owner, who finds the best fit mask. In some cases, suppressions can be more accurate than the intervals or vice versa, and this depends on the data type. Data owner needs to be accurate on choosing the best masking pattern. The masking is applied gradually, which depends on the value of ψ. This value provides the gradualist tool on applying masking pattern. The masking pattern criterion is chosen based on the probability multiplication. The results are approximate to the maximum sensitivity level as described below:

(3.8)

Let us consider the following examples of how the masking pattern is applied. In the first example, a user was given ψ=0.04, =4, while Q-ID= {Age, Sex, Suburb}. The probability for each Q-ID is shown in Table 3.4. The created semi-equivalent records are grouped based on the highest two Q-IDs, and these are P(age) and P(sex). Suppose that the chosen Suburb pattern is the taxonomy tree for Sydney suburbs or <Cutj>. The anonymisation process relies on grouping and masking. As explained before, the grouping of records is an iterated process. The first grouping stage filters out all the fully equivalent records, while the rest of the non-equivalent records are stored in SSG1 waiting for the next stage. In the next stage, the SSG1 group is grouped by the highest two probabilities, Sex and Age, while the Suburb Q-ID is masked by the taxonomy tree. The anonymised suburb should follow the sensitivity rule by finding a probable value ≤ ψ.

In Table 3.5, the first SSG1 record was anonymised by Suburb. Luckily, all suburbs in the first record were located in Penrith, hence, the generalized value became Penrith, since P(Penrith)= 1/40 = 0.025 < 0.04. In the second line, the number of grouped values was 8. Since the value of =4, then the masking can be applied on each 4 values separately. Thus, the first 4 suburbs are located in Mosman, while the rest of the 4 suburbs are mixed between different areas. Even three of them are in Mosman, but the forth suburb is from another area, so it has contaminated the masking level, and extra generalisation is needed. For this reason, the masking has moved to a higher level in the taxonomy tree, which is Sydney. Mosman value is accepted because P(Mosman)= 0.033 < 0.04. In the last record, the suburbs are located in one area, Randwick, but P(Randwick)= 0.47, which is > ψ . Therefore, the masking cannot be with Randwick, so Sydney masking is chosen. After applying the masking process on the SSG1 records. Finally, the anonymised data is stored in SG group.

Table ‎3.4- Three Q-ID attributes example

|  |  |  |
| --- | --- | --- |
| **Q-ID** | **P(Q-ID)** | **Description** |
| Age | 0.01 | From 1 to 100 |
| Sex | 0.5 | Male, Female |
| Suburb | 0.0001 | 850 suburbs in Sydney |

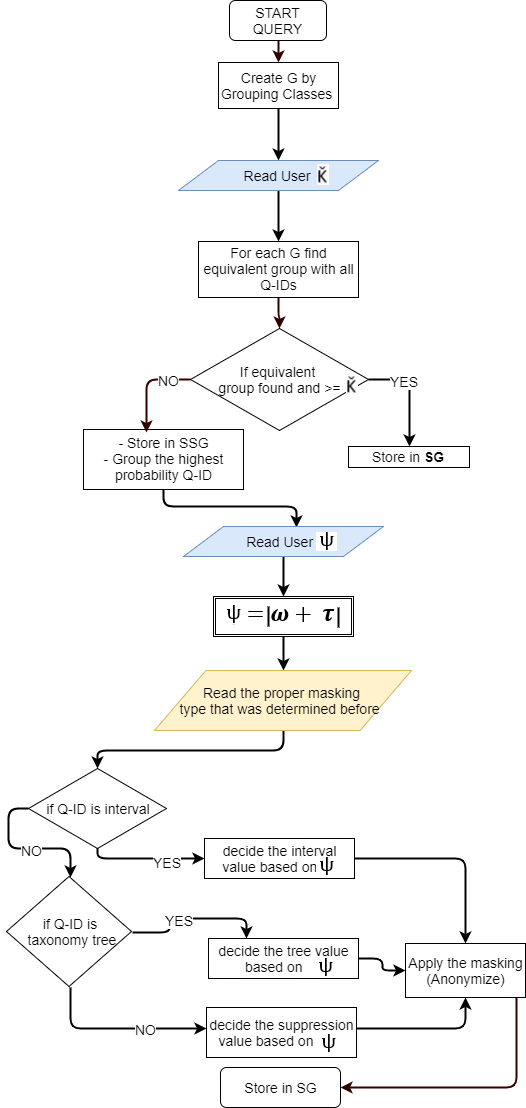


Figure ‎3.6- MDSBA algorithm

Table ‎3.5- Anonymisation for SSG1

|  |  |  |  |
| --- | --- | --- | --- |
| **Age** | **Sex** | **Suburb** | **Anonymised Suburb** |
| 25,25,25,25 | F,F,F,F | [Mulgoa](https://en.wikipedia.org/wiki/Mulgoa,_New_South_Wales), [Colyton](https://en.wikipedia.org/wiki/Colyton,_New_South_Wales), [Caddens](https://en.wikipedia.org/wiki/Caddens,_New_South_Wales),  Penrith | Penrith, Penrith, Penrith, Penrith |
| 30,30,30,30,30,30,30,30 | M,M,M,M,M,M,M,M | Mosman, [Balmoral](https://en.wikipedia.org/wiki/Balmoral,_New_South_Wales)  , Chowder, Obelisk, Parriwi , Taylors , Pearl , [Kirkham](https://en.wikipedia.org/wiki/Kirkham,_New_South_Wales) | Mosman, Mosman, Mosman, Mosman, Syd, Syd, Syd, Syd |
| 42,42,42,42,42 | F,F,F,F,.F | [Kensington](https://en.wikipedia.org/wiki/Kensington,_New_South_Wales), [Coogee](https://en.wikipedia.org/wiki/Coogee,_New_South_Wales), [Chifley](https://en.wikipedia.org/wiki/Chifley,_New_South_Wales), [Malabar](https://en.wikipedia.org/wiki/Malabar,_New_South_Wales), [Maroubra](https://en.wikipedia.org/wiki/Maroubra,_New_South_Wales) | Syd, Syd, Syd, Syd, Syd |

In the second grouping stage, the rest of the SSG1 records are grouped again with the highest Q-ID probability. This implies that records will be grouped by Sex only. The rest of the two Q-IDs, Age and Suburb, will be anonymised. The probability factor of both Q-IDs must not be above the ψ. Table 3.6 illustrates this kind of anonymisation. This grouping will result two records only, for male and female.

Table ‎3.6- Anonymisation for SSG1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Age** | **Anonymised Age** | **Sex** | **Suburb** | **Anonymised Suburb** |
| 29, 31,31,32,… | [25-35[,[25-35[, [25-35[,[25-35[ | M, M, M, M,… | [Lansvale](https://en.wikipedia.org/wiki/Lansvale,_New_South_Wales), [Cabramatta](https://en.wikipedia.org/wiki/Cabramatta,_New_South_Wales), [Carramar](https://en.wikipedia.org/wiki/Carramar,_New_South_Wales),  [Yennora](https://en.wikipedia.org/wiki/Yennora,_New_South_Wales),… | Fairfield, Fairfield, Fairfield, Fairfield,… |
| 22,25,32,32,41,… | [20-35[, [20-35[, [20-35[, [20-35[, [40-45[,… | F, F, F, F, F,… | [Kensington](https://en.wikipedia.org/wiki/Kensington,_New_South_Wales), [Coogee](https://en.wikipedia.org/wiki/Coogee,_New_South_Wales), [Chifley](https://en.wikipedia.org/wiki/Chifley,_New_South_Wales), [Malabar](https://en.wikipedia.org/wiki/Malabar,_New_South_Wales), [Maroubra](https://en.wikipedia.org/wiki/Maroubra,_New_South_Wales),.. | Randwick, Randwick, Randwick, Randwick, Randwick,… |

As noticed in Table 3.6, the anonymised age is masked by intervals, and the table shows only a small part of data. For security purposes, the minimum interval is set to 5. The probability factor for the first record is P(total)=P(Fairfield) \* P(10 years age interval)= 0.037 \* 0.1= 0.0037, which is smaller than ψ. The interval was chosen based on the Age values. The first value of the Age record was 29, and the forth value was 32. To fulfil *k-anonymity* requirements, a minimum of 4 records must be equivalent. Thus, the interval has included the first four values. In the second record, the Suburb masking of Randwick was not possible in the previous table. However, in this stage the factor of both Q-IDs has enabled a better masking level in the Suburb attribute. The comparison between Table 3.5 and 3.6 illustrates the difference between both Suburb records. In the first table, the Suburb anonymity was generalized to (Sydney), while in the second table the generalising was set to (Randwick). This is related to the probability factor decrease.

It is clear that each record of the previous Table 3.6 contains a long array of values. This is due to the Sex grouping. This kind of grouping separates all the dataset into two compressed records only. Hence, anonymising each one of these arrays is not an easy task to tackle. Here are some facts we need to remember before masking this array. The masking pattern is chosen based on the sensitivity value ψ. The algorithm can be used to determine the closest masking pattern. The general algorithm is illustrated Figure 3.6.

Anonymisation methods apply data distortion using masking operations. Masking implies a taxonomy tree, suppression or discretization. The taxonomy tree is the key anatomy for data masking. It implies hiding special information by generalising them. For example, if the data contains person’s suburb as ‘Sydney’, then the taxonomy tree contains Australia (country) 🡪 NSW (state)🡪 Sydney (city). In Bottom-Up Generalization, the masking of the first cut is NSW, and the second cut is Australia. Discretization means replacing numerical values with a single interval and is denoted by *Int.* The interval deals with numerical data, where a set of numbers is presented by two numerical values for start and end. Finally, suppression of value means replacing all its relevant values with the sign \* or other characters. This operator is denoted by ‘Sup’ [99]. Also, suppression can be presented by other values, such as ‘any, or ‘person’. Some data can be only masked by suppression, such as person’s gender {Male, Female}. Other data can be only anonymised by using interval or taxonomy tree, as shown in Table 3.7. The three masking patterns are presented as <∪Cutj ,∪Supj ,∪Intj> [26]. The sensitivity value of the attribute S is calculated based on the user access level and other factors. This sensitivity value is the milestone that determines how the masking pattern will be applied on Q-IDs. Firstly, the Q-ID data type is chosen, as shown in Table 3.7. Secondly, the masking pattern is being implemented on the chosen data type. This imposes three masking tools for anonymisation, which are; taxonomy tree level for ∪Cutj pattern, interval distance for ∪Intj, and the number of suppressed digits for ∪Supj pattern.

Table ‎3.7- The masking Pattern for some data types

|  |  |
| --- | --- |
| **Masking Pattern** | **Data Type** |
| ∪Cutj, ∪Intj, ∪Supj | Date |
| ∪Intj, ∪Supj | Integer |
| ∪Cutj | Polynomial *(Ex. Education tree)* |
| ∪Supj | Binomial *(Ex. Yes / No)* |
| ∪Intj | Real |
| ∪Supj | Text |

The user access level influences the chosen masking pattern for Q-ID. In Table 3.7, some data types can be distorted by any of the three masking patterns while others permit one or two patterns only. For example; postcode can be masked by either one of the three patterns while gender can only be suppressed by ∪Supj. Also, masking methods imply different security levels, as explained in the next section. Two main types of masking are introduced in MDSBA, these are

#### Taxonomy Tree Masking Example

On grouping records, a long array is created containing a large bag of data records. If the array was a set of text values that require ∪Cut or taxonomy tree. Then the algorithm reads the masking patter, and applies the masking accoringly.

Referring to Table 3.5 example, if the array objects for Suburb is arr1={[Roselea](https://en.wikipedia.org/w/index.php?title=Roselea,_New_South_Wales&action=edit&redlink=1), Mosman, [Balmoral](https://en.wikipedia.org/wiki/Balmoral,_New_South_Wales) , Chowder, Obelisk, [Cowan](https://en.wikipedia.org/wiki/Cowan,_New_South_Wales), Parriwi , Taylors, Macquarie}. Generalizing the array objects to the first taxonomy level will result in the following array arra1= {Hornsby, Mosman, Mosman, Mosman, Mosman, Hornsby, Mosman, Mosman, Ryde}. Next, the array is sorted based on count of objects appearances as arr1= {Mosman, Mosman, Mosman, Mosman, Mosman, Mosman, Hornsby, Hornsby, Ryde}. The array objects are now ready for equivalency check. The check reads a number of objects, checks the equivalency and stores them in arr2. Here in this example, the first group in arr2 is arr2= {Mosman, Mosman, Mosman, Mosman}, while the second group contains 5 objects and not 4. If reached the last object in arr1, then read all the remaining objects. Hence, if the number of the last objects, then all objects are read and generalized. The five objects are {Mosman, Mosman, Hornsby, Hornsby, Ryde}, since the objects are non-equivalent, then the generalisation moves to the next taxonomy level. The next level is given by Sydney, so the generalisation will be {Syd, Syd, Syd, Syd, Syd}. The final array will contain the following objects arr2= {Mosman, Mosman, Mosman, Mosman, Syd, Syd, Syd, Syd, Syd}. The taxonomy tree anonymisation algorithm is further explained in section 4.3.2.3.

#### Discretization Masking Example

The discretization or interval is another masking pattern that is implemented in MDSBA. This masking pattern can be more accurate than the taxonomy tree. However, choosing the masking pattern is related to the data type. Data with real, integer, or date can be anonymised by intervals. The interval requires a range of minimum (*Vmin*) and maximum (*Vmax*) values to generalize numbers within the range. The interval range can be derived from the ψ value. The interval is calculated by ∪Int =1 / ψ, where UInt ≥ 5. So if the interval was smaller than 5, then it will be set to 5. Moreover, the minimum and maximum intervals are chosen with units of 5 starting from zero, and base-five units are added on each time extension. Assigning the interval in this way is essential to protect data privacy, especially data on both interval ends. To explain this, let us consider two values of {13,14} and an interval range= 2 . The interval can be written as [13-15[, which clearly tells the adversary about the original numbers. The idea behind creating a range of units is hiding the original numbers from adversaries.

The algorithm is very similar to the previous taxonomy tree masking. The main essential part is calculating the interval for each group of array objects. It is apparent that masking process must start with taxonomy tree masking first. If masking requires more than one Q-ID, then the first Q-ID masking must be the taxonomy tree, then the interval and finally the suppression. The process concludes three main IF statements to determine the best interval for each group of objects in the array. The numerical array is firstly sorted. The Q-IDs factor is calculated and compared with ψ. If the Q-ID factor ≤ ψ, then the algorithm proceeds with the masking. The range is calculated by, so the . Eventually, we are able to find the minimum and maximum values of the interval, by referring the small array object to the low unit of 5, and adding the found interval to the minimum value, to find out the minimum value. For instance, if the minimum value is 6 and the interval is 7, then the *Vmin* =5, and the *Vmax* =15, as a result of (5 + 7=12). It is clear that both minimum and maximum values must refer to base-five rule.

After finding the interval, the algorithm loops for all objects in the array. The first IF statement finds the all objects between the minimum and the maximum values. The second IF statement is only used when the object are the last in the array. The third IF statement verifies whether the number of objects within the interval fulfil the value. The last forth IF statement creates the intervals. To understand the anonymisation algorithms, few more examples are illustrated in section 4.3.2.4.

## Summary

This chapter reviewed a large part of MDSBA. This included general definitions, grouping and probability methods. In this chapter, it was proven that that data equivalency increases parallel with the general growth of data. This induced a need for re-considering big data anonymity techniques. Four characters were identified for any developed big anonymity method. These characters included; equivalency increase, information gain and security, parallel algorithm, and gradual access. It was explained that sensitivity-based anonymisation method carried on the big data characters. This novel method is able to provide an access control for multiple level of user’s access. The main concept of MDSBA is dividing datasets into small groups of attributes (vertically), and equivalent groups of records (horizontally). These two types of grouping helps parallelization of data operations over multiple nodes. Big data needs to operate in a large scale framework such as Spark or Hadoop. These frameworks require special algorithms that are able to cope with the framework structure. Two main masking tools were discussed; taxonomy tree and discretization. The operation sequence assigns the taxonomy tree filtration first, and then discretization or interval.

The next chapter implements practical solutions, provided with experiments. MDSBA will be further examined and compared with the other popular anonymisation methods. Hadoop framework and ecosystems such as; Pig and Hive will be applied with several scripts to examine the efficiency and performance of anonymising data by MDSBA.

# - Implementing Sensitivity-Based Anonymisation 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 anonymisation 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 [97]. 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 [36].

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 [100]. Recently developed methods in *k-anonymity* moved toward finding parallelization techniques in the anonymisation 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 randomly such as; Two-phase Multi-Dimensional Top-Down Specialization method TPMDTDS [26] and the method in [101].

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 anonymisation 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. Section 4.2 describes 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 anonymisation methods were related to the first version of Hadoop [48].

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 [102].

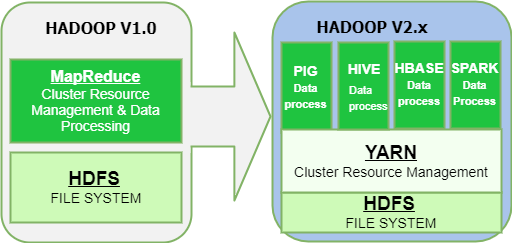


Figure ‎4.1- Comparison between Hadoop v1 and v2

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 [103].

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 [104].

### 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 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. It boosts Hadoop to incumbent any new technologies found within the data centre, so they can take advantages of cost-effective, linear-scale storage and processing [105].

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 analysers. The preparation implies data anonymisation as requested by data owners. Data analysers are then ready to access the anonymised data for statistical analytics. Datasets can be archived or live data, hence, anonymising archived data technique is different from live data. Next chapters will focus on archived data, while the live data will be discussed in chapter 7 [106].

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, UDF can be embedded in the script to provide powerful programming algorithms [107].

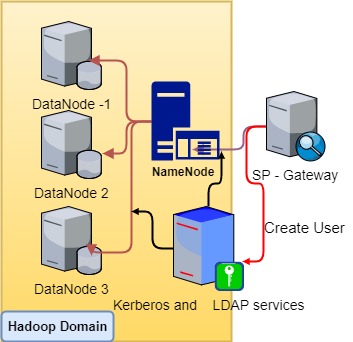
## Hadoop security

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 a Lightweight Directory Access Protocol (LDAP) or Network Inform Service (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

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 NIS, LDAP, DNS or BIND [108]. 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 [109].

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 [110].

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 [111].

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 three 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 [112].

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 ecosystems, 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 [113]:

Create service principals for YARN, HDFS, and mapred.

Create keytabs for the three services.

Copy keytabs to Hadoop configuration folder.

Update Hadoop XML configuration files with the keytabs locations.

Start Hadoop.



Figure ‎4.3- Kerberos general steps to secure services by Administrators

## Deploying Sensitivity-Based Anonymisation by using Hadoop Ecosystems

### Pig 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 [114].

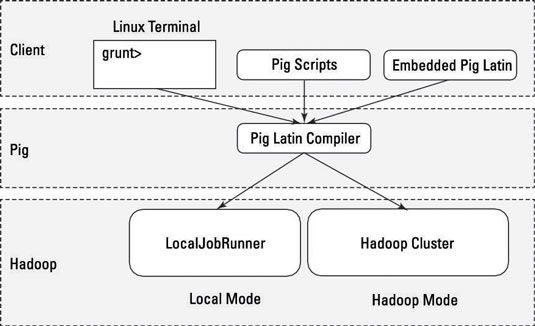
Traditionally, SQL language is the database dominator to manage and alter data. Hive is the data warehouse system for Hadoop, which aims to simplify Hadoop usage for data workers by providing the SQL-like language for Hadoop [115]. Pig Latin is another Hadoop tool that manages warehouse system, by using a proprietary scripting language. Pig Latin treats data as a set of tuples, which fosters tackling very large data sets. Thereby, substantial parallelism and a slew of optimization techniques are supported. Pig provides customized program for UDF, by supporting many common languages such as Java, Python [116], JavaScript, Ruby [117] or Groovy [118]. Similar to Hive, Pig supports ad-hoc queries, joins, and other SQL-like operations [119]. Pig compiler resides at the top of YARN and HDFS, which results in the client being responsible for running the script. Pig Latin is a combination notation of SQL-like and Java idiom. It allows three modes of user interaction; Interactive, batch, and embedded mode. In the interactive mode, the client is presented with an interactive shell, called Grunt. The interface allows users to interact commands line by line. This mode is suitable for developers and debuggers. The batch mode is the production mode, where users write a complete code and stores it in a file with an extension of (.pig). Finally, in the embedded mode, Pig is provided within Java library, by writing a Java code, and calling Pig library from inside Java [120].

Pig controls the data flow of tuples by creating data bags and maps. The bags are aggregated tuples of potentially varying structures, which may contain duplicates. Pig Latin was designed based on Functional Reactive Programming (FRP). The FRP depends on lazy evaluation and push/pull model. Pig was designed by pull or (iterator) model through the execution pipeline. The pull model was chosen over the push model for many considerations, such as UDF, and bags nested inside tuples [121]. Pig operates in a similar way to hadoop, which includes; map, sort, combine, shuffle, merge, and reduce. Data is aggregated by tuples to conclude bags, which leverages execution performance and speed. Pig Latin operator is triggered by either (DUMP) or (STORE), known by evaluation. Without a (STORE) command, the operator does not execute any task as a lazy evaluation. This has an advantage in the logical plan, since this may optimize the program structure by using Directed Acyclic Graphs (DAG) [122]. Moreover, if more than one STORE command in the script, then data is split and multiplexed, so they are processed in parallel. The SPLIT operation maintains a one tuple input buffer for each sub-flow or split [96].

One of the technical difficulties that may face developers of big data models is the JVM limited heap memory. JVM developers recommend a maximum of 25% memory allocation of the total RAM, which causes a memory waste and Java Heap memory errors [123]. In MapReduce structure, created JVM containers may consume the major size of the memory, which reduces the Java Heap memory errors. Therefore, it is more efficient to implement JVM through MapReduce, rather than using locally created JVMs. Since UDF adopts local JVMs, then it can be considered as a bottle nick in MapReduce. Avoiding a large data size flowing to UDF is essential to reduce error rates by Java Heap memory failure. However, this does not totally prevent operations failure. When using Pig, there is a possibility of passing a large size of data bag for materializing data to database format. In the usual case, Pig Latin is able to cope with the large size of bags, by implementing (combine) between tuples. However, memory overflow may appear in Pig due to materialization of large bags of tuples between and inside operators. In some cases, Pig needs to materialize large bags inside the pipeline for holistic bag computation. For this reason, another technique of spill arise to avoid the over-flow. This technique transfers some data rows to the disk in a temporary location. In few cases, the spill may fail to manage the large data flow, which causes an error of (out-of-memory) exemption. such an error can be avoided by increasing the JVM heap memory, or modifying the Pig Latin script, or even both actions [51].

Pig is still under development and immature solution. A considerable number of concerns have not been resolved yet. Pig misses out an optimized storage structures like indexes and column groups. Also, Pig shell needs a considerable time to start, and to clean up jobs. It was developed merely for batch processing, therefore, it is not efficient for data stream. Pig Latin language is inefficient to complete some programming algorithms, such as iteration, nested iteration, and if statements. Therefore, developers need to implement many UDF programs. In the UDF, error messages are general and not clear. Developers agree that the biggest advantage of Pig is simplicity of coding Pig Latin script, and the smooth logic of data flow [124]. Unlike the coding complexity in Spark, as discussed next.

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 [125].



*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 [36]. 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 [100].

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].

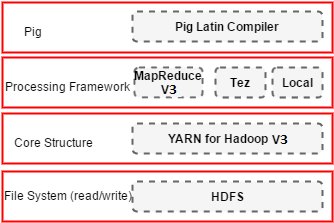


Figure ‎4.5-Pig structure at the top of the processing framework

MDSBA is proposed mainly for the MapReduce structure. MDSBA divides the anonymisation 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.

### Anonymisation 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 anonymise data before permitting user’s access. The script is generated based on some XML files received from the federation service or a 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 framework is described in the next chapter. Pig algorithm relies on vertical and horizontal grouping on applying anonymisation. 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 anonymisation process. Each group is mapped to one or more of business role(s). On the other hand, vertical grouping is applied during the anonymisation 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.

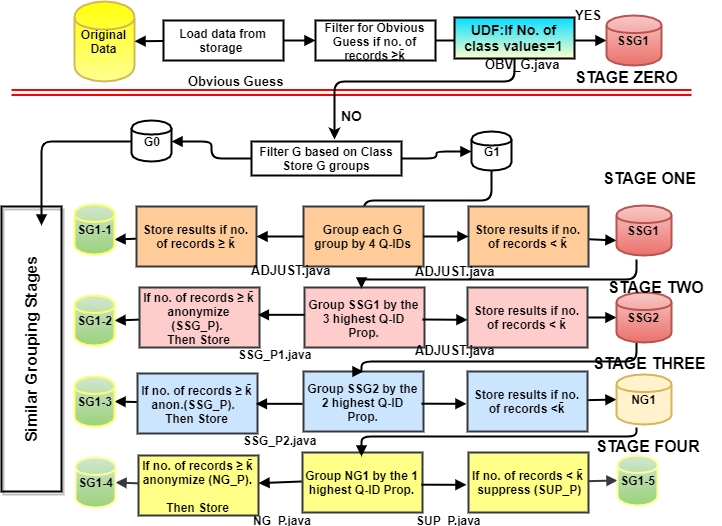
This section focuses on implementing vertical grouping by Pig Latin script. The vertical grouping is the core concept of data anonymisation in MDSBA. Pig Latin algorithm can be divided into four major procedures: filtering, grouping, ungrouping, and masking. The first two procedures are accomplished 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 anonymisation is dividing anonymisation 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 anonymise four Q-IDs group*

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 anonymised. The anonymisation 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 anonymisation.

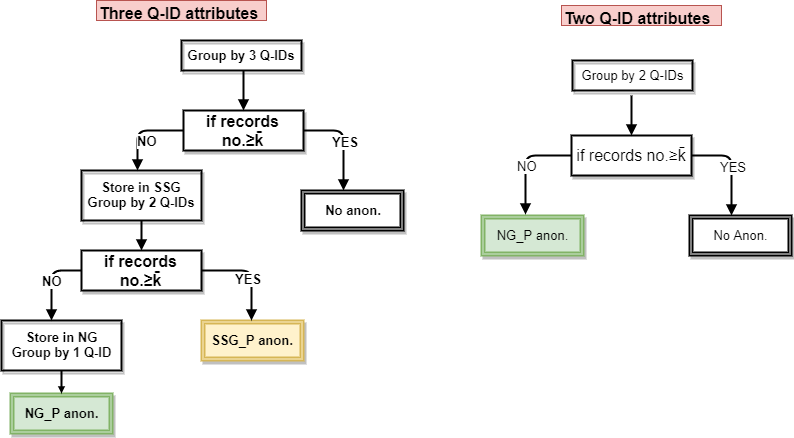


Figure ‎4.7-A. Anonymisation process of three Q-IDs. Figure 4.7-B. Anonymisation process for two QIDs

The grouping command is implemented for the highest probability values of two Q-ID attributes, while the lowest two probability values are anonymised. 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[114]. 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 anonymisation will take a place. Stage two filters three Q-ID attributes of SSG1 data, so if cnt ≥, then SSG\_P1 Java program will anonymise 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 anonymised by SSG\_P2. In the final fifth stage, records are grouped by the highest QID probability and anonymised 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, every G group produces SG groups including, SG1-1, SG1-2…... SG1-5. The SG groups are collected and merged together in one file. 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

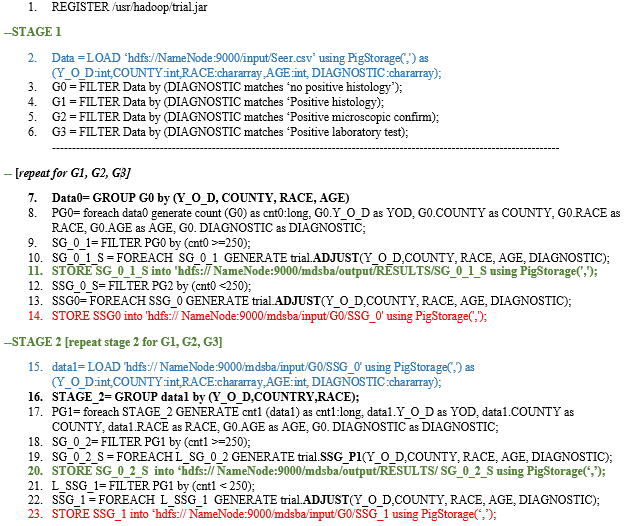
Let us study the following complete example of anonymising data by implementing Pig Latin script, as shown in Figure 4.8. The anonymisation is applied to Seer Cancer Data. Table 4.1 shows the suggested Q-IDs, and the probability for each attribute. Table 4.2 shows the Class attributes with four sensitive values.

*Table ‎4.1- 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 ‎4.2- Seer Cancer Data Class (four sensitive values)

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



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

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 4.1, 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 anonymised with a sensitivity level ψ=ω+ τ=0.124-0.02≈0.122. This sensitivity level indicates anonymisation degree that will be applied on attributes. For instance, if four numerical Q-ID attributes need to be anonymised, 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 anonymisation 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 anonymised 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 4.8, 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 4.2. The filtration commands are shown in Figure 4.8 / 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 anonymisation 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 anonymised by SSG\_P2, and stored in SG location. The final stage groups (RACE), and anonymise 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. In the Seer data example, the script generates four G groups, as described in Table 4.2. The more created G groups, will lead to a better distribution of data in parallel. Table 4.2 is created and prepared before conducting any anonymisation 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

In UDF, all registered operations are executed in a black-box [126]. 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 thoroughly studied, in order to mitigate the large data flow impact.

In MDSBA, 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 anonymise 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 anonymise 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 anonymising 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 4.3 describes the Java programs needed for each G(QID) group.

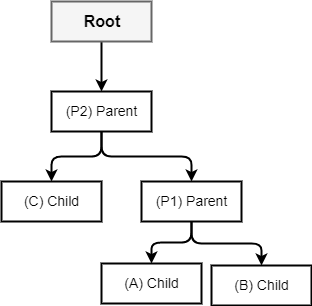
Table ‎4.3- Illustrates the needed programs for each Q-ID group

|  |  |
| --- | --- |
| **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 anonymisation tasks. Algorithms that mask taxonomy trees or intervals the major two algorithms among the six program.

#### Taxonomy Tree Anonymisation 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 anonymisation 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 4.9.



*Figure ‎4.9- Taxonomy tree structure for XML files*

Figure 4.9 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 4.10.

*Figure ‎4.10- Example of cars taxonomy tree*.

Figure 4.10 shows an example of car’s taxonomy tree with the probability for each node. Referring to Figure 4.9 and branches, an XML file can be generated as illustrated in Figure 4.11. 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 4.10 taxonomy. On anonymising any value, the algorithm fetches all children nodes to find out the match word. If the anonymisation 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 anonymisation 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 anonymisation 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 anonymised by the taxonomy tree. The algorithm anonymises all objects with parent 1, and checks the results. The results verification tends to count the number of equivalent objects after the first anonymisation 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 anonymised to the next parent. Once again the equivalent records are verified against the value of. The algorithm is shown in Figure 4.12.



*Figure ‎4.11- XML file example for the taxonomy tree in Figure 4.10*

The algorithm verifies the number of objects and compares the number with. The algorithm is divided into two main sections; initial anonymisation, and complete anonymisation. The initial anonymisation imposes a complete anonymisation for each array object. The reading is pre-conditioned by comparison between each parent’s probability and the value of ψ. The initial anonymisation is essential to accomplish most of the anonymisation process. This may save a considerable time, if the data does not need any extra anonymisation. The complete anonymisation 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  //Anonymise all array objects to parent1 or higher (Initial Anonymisation)  **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 anonymised 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 anonymising 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  //anonymise to a higher parent  IF parent ≤ parents\_no  parent\_value=array1(i-1).readXML(parent)  //Update the new anonymised 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) // Anonymise by the root  //Update the new anonymised 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 ‎4.12- Taxonomy masking algorithm

The algorithm can be summarized in the following steps:

* + - 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 anonymising 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 anonymise to the next parent
      8. Continue scanning until there is no equivalent records smaller than

The UDF program anonymises the taxonomy tree values, and concatenates them with the rest of the attributes. The program inputs the data bag with compressed tuples and returns anonymised tuples to the Pig program. For a better understanding of the UDF anonymisation procedures, let us consider a substance of Seer data as shown in Table 4.1. 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 ‎4.4- 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 anonymisation 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 anonymisation 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 anonymised 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 anonymised. 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, generalising to a higher parent did not resolve the dissimilarity. Therefore, a higher generalisation 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 generalisation is County (Root)={USA State, USA State, USA State, USA State}.

#### Interval Anonymisation Algorithm

MDSBA implements UDF in different locations. This is essential for two main purposes; anonymising and ungrouping. In anonymising, three masking types of interval, taxonomy tree, and suppression are implemented. Figure 4.13 shows the algorithm for anonymising any numerical group. In the following algorithm, one numerical type attribute is used for anonymisation. 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.

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 anonymise data with the minimal number of loops.

|  |
| --- |
| 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 anonymise 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 ‎4.13- Algorithm illustrates the numerical values anonymisation

To understand the anonymisation 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 anonymises 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 4.13, the algorithm includes four sections. The algorithm, 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 anonymisation and Other Methods in Big Data

The first experiments are conducted on the Adult and Seer Datasets from the UCI Machine Learning Repository [88]. The data are public benchmarks for anonymisation 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 anonymisation methods. Testing the information gain in any anonymised 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 4.5. 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 4.6-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 4.6-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 4.6-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 ‎4.5- 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 ‎4.6-A Adult data after generalising EDU Table 4.6-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 anonymisation, and predetermining the best cut, before anonymisation, 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 anonymisation 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 anonymisation loss.

### UDF algorithm for BUG

The applied algorithm in this UDF was derived from the advanced BUG (Adv-BUG) introduced in [79]. The procedures are close similar to the other BUG [127] and MRBUG [24] methods. Adv-BUG deliberately splits a random number of data groups, and then performs generalisation 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 generalisation applies Parent1 for each Q-ID attribute separately. For instance, in Adult data attributes (EDU, SEX, AGE), the algorithm applies generalisation to EDU followed by InfoGain calculations. Next, the algorithm applies the generalisation on SEX followed by InfoGain calculations. Finally, AGE is generalized and InfoGain is calculated and compared with the other two results. The accepted generalisation among the three attributes is the one with the highest InfoGain result. In case if Parent1 generalisation does not satisfy *k-anonymity*, the iteration moves generalisation 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, anonymisation level AL0, anonymity parameter *k*. |
| **Process:**  1. Scan data set D.  2. Initialise generalisation 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 generalisation set AGSet out of all the active generalisation candidates  6. Choose the generalisation 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 generalisation set AGSet out of all the active generalisation candidates  12. Choose the generalisation for the attribute with the highest InfoGain |

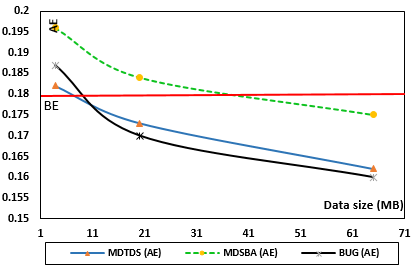
*Figure ‎4.14- The BUG driver algorithm implemented in UDF*

### Small Data Size experiments

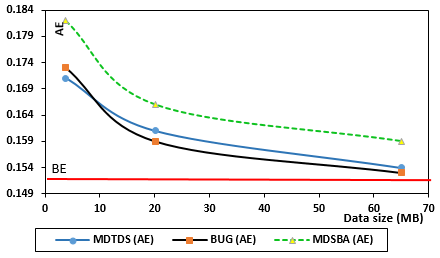
The first experiment aimed to compare between MDSBA BUG, and MDTDS in the traditional dataset. The used Adult and Seer datasets were divided into training and testing dataset. The chosen datasets were based on 70-30 ratio, where 70% of the source dataset for training model, and 30% of the source dataset 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 4.1 & 4.2. Adult data with Q-IDs= {Age, Edu, Sex} and the class contained 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 anonymisation methods. The first comparison relies on Naïve Bayesian classifier. The anonymisation was applied to two datasets, Adult and Seer datasets. After completing the anonymisation processes for each dataset, the classification error was calculated by using RapidMiner Studio. The calculation was conducted before and after the anonymisation. The experiment followed similar steps as in [23]. 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 4.15 and 4.16 show the classification error comparison between the three anonymisation methods. The comparison considered *k*=15 in all trials. Three different data sizes were compared in the three anonymisation methods.



*Figure ‎4.15- Classification error for three sizes of Adult datasets*



*Figure ‎4.16- Classification error for three sizes of Seer datasets*

In Figures 4.15 and 4.16, 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 anonymisation impact, but computationally expensive. BUG also, follows similar techniques. ILPG is calculated on each iteration time, and generalisation is given according to InfoGain scores. On the other hand, MDSBA, pre-determines the anonymised Q-ID attribute, and pre-calculates the amount of anonymisation applied. Hence, there is no need to iterate the program to make a decision, since the anonymised Q-ID is pre-determined, and the minimum generalisation level is pre-calculated.

Figure 4.15 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 anonymisation. 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 anonymisation loss in various anonymisation 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 anonymising data by either MDTDS or BUG.

However, MDSBA computation cost is lower than BUG and MDTDS. Figure 4.17 illustrates the processing time for Seer cancer data. The diagram shows a comparison between the three methods performances 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.

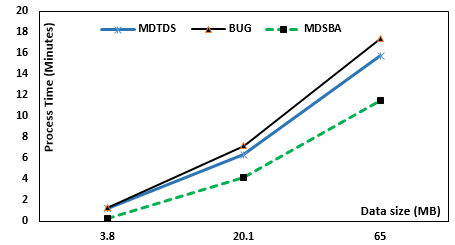


Figure ‎4.17- Processing time comparison for the three anonymisation methods

### 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, Adult data, Heart Disease data as illustrated in Table 4.7, and Kasandr data as illustrated in Table 4.8. 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 anonymisation process. The purchaseDate is generalized by a taxonomy tree of day, month, and year, as shown in Figure 4.18.

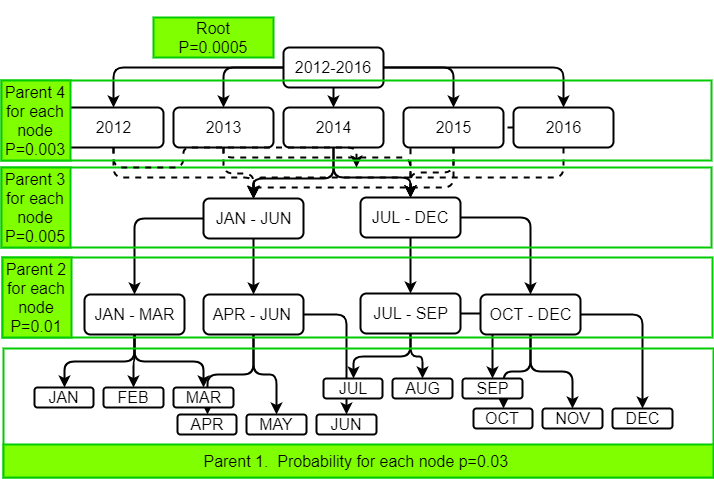
Table ‎4.7- Heart Disease Q-ID attributes, propabilities and the class values

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 ‎4.8- 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 4.18] | * 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 4.9 and Table 4.10. 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 4.9. 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 4.10, 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 ‎4.18- Taxonomy tree for purchase Date attribute in Kasandr dataset*.

Table ‎4.9- 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 ‎4.10- 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 anonymisation methods. Moreover, more datasets have been added to a better evaluation. The calculation was conducted before and after the anonymisation. Both BE and AE were measured for each dataset. The BE values are as follow: for Adult dataset 0.18, for Seer 0.15, for Heart Disease 0.16, and for Kasandr 0.17. Figures 4.19 - 4.22 show the classification error comparison between the three anonymisation methods. The comparison considered *k*=50 in all trials. Three different data sizes were compared in the three anonymisation methods.

All datasets show similar results regarding AE values. The anonymisation was conducted based on splitting data in BUG and MDTDS as shown in Table 4.9, and Table 4.10 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 4.19 and 4.20 showed 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 GB does not require many splits in the three anonymisation 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 the big data size. Fortunately, the information gained, after the anonymisation with MDSBA, increases parallel with the data size increase. When data size is bigger than 1.2 GB, then Figures 4.19 - 4.22 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.

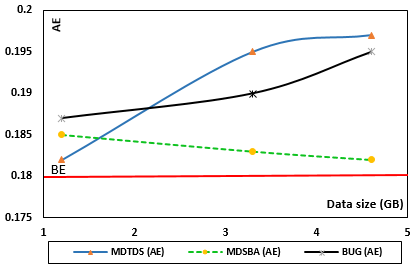


Figure ‎4.19- AE in Adult dataset results for three different methods

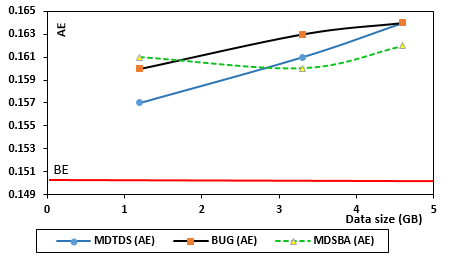


Figure ‎4.20- AE results in Seer dataset for three different methods

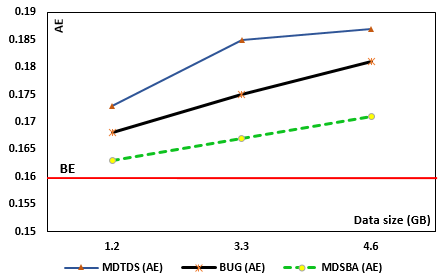


Figure ‎4.21- AE results in Heart Disease dataset for three different methods

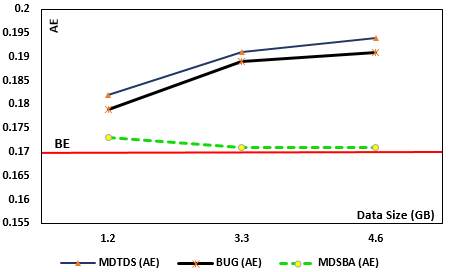


Figure ‎4.22- AE results in Kasandr dataset for three different methods

## Anonymisation Classification

Many techniques have been implemented to find a benchmark for measuring the performance of the anonymisation methods. Techniques, usually compare data before and after anonymisation to ascertain the information loss. The most widely used classifiers are Naïve Bayesian, C4.5, and K means [128]. The classifiers present algorithms to predict output results as per classified data. The comparison between pre-anonymisation and post anonymisation 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 anonymisation. Different classifiers may output different classification error. Hence, classifiers measure the prediction level rather than the actual anonymisation level and information usefulness. In some previous experiments, the classification error of AE was lower than BE level, which is inadequate. For instance, Figure 4.15 showed a drift of AE below the BE value. The BE value presents data before anonymisation, while AE presents anonymised data. Various anonymisation 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 anonymisation, denoted by Disruption (Ɗ). The disruption value is a benchmark that gives a general indication of the size of anonymisation loss. As shown in Equations 4.1 and 9, each anonymised block of tuples is calculated individually, and finally, the Ɗ value is the result of the total summation of all anonymised blocks. Each block of data is a data bag produced by grouping a set of tuples. Let us assume that an anonymised block of data contains some M records, in a total number of N dataset records.

Ɗs ‎4.1

Ɗ[total] ‎4.2

Equation 4.1 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 anonymisation. Hence, anonymising three Q-IDs will result in a higher Ɗ value than anonymising two Q-IDs. Each block must have similar attribute values. If a block was found with two different anonymised values, then it will be split into more than one block. The following example illustrates Equations 4.1 and 4.2. Recalling the adult data, and considering the total number of records is 500. Two blocks of data were anonymised by two Q-IDs of Education and Age. The number of anonymised records for these two blocks is 3 of each. The education anonymisation was given parent 2, which relates to (certificate) and (degree), respectively. Both anonymised blocks are shown in Table 4.11. 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 4.2, the total value of Ɗ = 0.025.

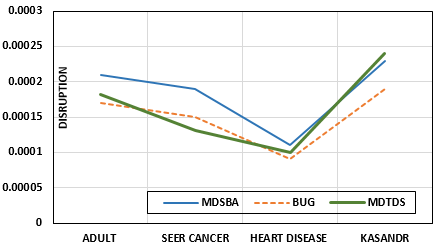
Table ‎4.11- Disruption example with Adult data

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 anonymisation 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 Anonymisation 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 anonymisation 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 4.23 showed 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 4.24, showed 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 ‎4.23- Disruption comparison for =4 and the 4 datasets*

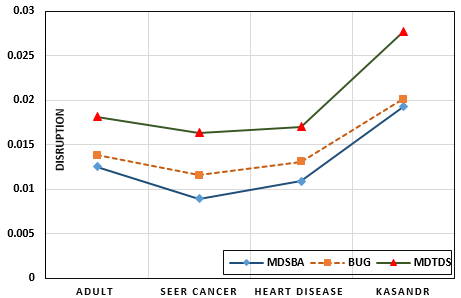


Figure ‎4.24- Disruption comparison for =50 and the 4 datasets

These results are expected, since the smaller value of *k*=4 does not require any split in BUG or MDTDS. It was examined earlier that BUG and MDTDS gain better results in information usefulness, if no split has occurred. Therefore, Figure 4.23 showed a higher disruption value for MDSBA, while Figure 4.24 showed 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 anonymisation level in MDSBA on the variation of. Figure 4.25 showed 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, anonymisation was applied on verities of datasets, and it was found that anonymising 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.

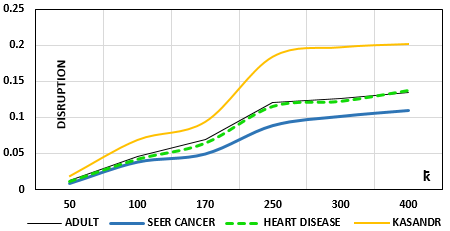


Figure ‎4.25- The relationship between Ɗ and variation in MDSBA.

## Performance Comparison

Figure 4.26 illustrated a considerable time difference in processing the three anonymisation methods. It showed the lowest processing time for MDSBA, followed by MDTDS, and finally BUG. The more accurate data after anonymisation 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 Giga’s was prepared and tested for processing time. The Seer Cancer datasets used in this experiment are: 1.2 GB, 3.3GB, and 4.6 GB. The experiment has measured the performance of three different data sizes, which showed that MDSBA had the lowest processing time. The disparity of the processing time was small between MDSBA and the rest of the methods, when datasets was small. The disparity has increased with the increased sized of data.

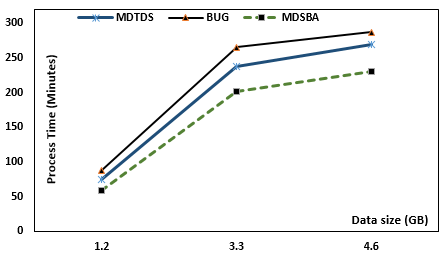


Figure ‎4.26- Processing time comparison for the three-anonymisation methods

The second part of the experiment showed a good performance for MDSBA. Figure 4.27 illustrates the decline of the three data sizes on anonymising 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.

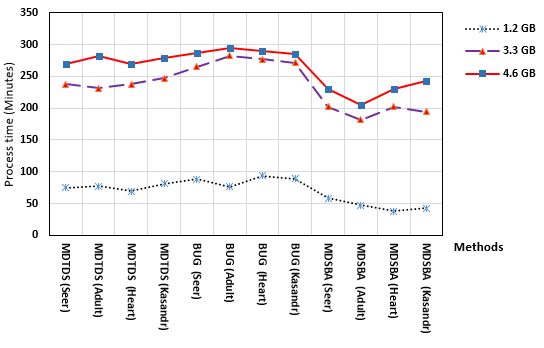


Figure ‎4.27- Processing time comparison of various datasets and sizes

## Summary

In this chapter, several experiments are conducted to measure and compare the MDSBA method. The experiments showed promising results in anonymising 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 anonymisation 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 anonymisation parameters. In MDSBA, the anonymised Q-ID and the amount of masking are computed prior the anonymisation process. Moreover, the three anonymisation 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 anonymisation 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 anonymisation. 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 anonymisation. 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 anonymisation.

# - Framework for Sensitivity-Based Anonymisation

Hadoop security concurrent many modules of protection. Hadoop File Distributed System (HDFS) contains sensitive information that needs to comply with owner’s security policy, to compromise a granular access for multi-user. Multi-security levels must be available to protect the organization assets of any security breach by securing; HDFS access, data analytics processes, Hadoop domain, data, and processes/files of the operating system. Big Data ecosystem is prone to multiple user’s access who request data analytics. For instance, organization’s alliance of business partners, contractors and sub-contractors may require accessing data for data analytics. Also, each organization may conclude more than one department such as human resources, financial, engineering or medical departments. This imposes a multi-layer security system to reduce data breach [113].

Hadoop ecosystems operate with tools reside at the top of Hadoop core such as Sqoop, Flume, Pig, and Hive. These tools are essential for big data management and operations. The tools can 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; HDFS, YARN, and JobHistory. The three parts can be secured by using Authentication, known by Hadoop secure mode, so each user and service needs to be authenticated by Kerberos before using Hadoop services, as discussed before in section 4.2.

In data analytics, current Hadoop security features can provide a satisfactory level of protection if it was implemented and configured properly. In fact, the risk may appear 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 large number of users, which in turn increases the percentage of exploring prohibited attributes. We need a framework that can organize and control the amount of accessed data, by implementing the anonymisation method. Hadoop security features are unable to provide such a level of security to data. Hence, Hadoop security features secure data from any unauthorized access only and without providing a robust access control.

Sensitivity-Based Anonymisation framework provides a security access control that mimics Role-Base Access Control (RBAC). The framework provides a granular security access for multi-user levels. This is accomplished by providing an anonymisation in a fine-grained access control. Data owners may wish to provide organizations with verities of access levels. Also, each organization may contain several departments with several data interests. For instance, human resource departments may request patient’s private data such as names and contact details, while researchers may request patient’s age, sex, and health status. MDSBA framework resides in two different locations; Federation Service (FS), and Service Providers (SP). The data is managed and controlled by Data Owners (DO). The DO is considered to be trusted, while the data analyst id not trusted.

In order to implement MDSBA in a fine-grained access control, several services should be able to map between FS and SP. The mapping should match between the assigned access policy on FS side, and the access control policy on SP side. The mapping concludes three different services cooperate interactively to transfer the given access permissions from the FS to the SP. The services are: core on the FS side, initialiser, and Anonymiser on the SP side. The core service is presented by applications that may operate at the top of the federation service/single sign-on. The core service contains of a database that stores the details of organization’s access levels, and general security values. Before delving in these three services, it is essential to understand the transmission method between SP and FS sides.

## Security Assertion Markup Language (SAML)

SAML is an XML base single sign-on (SSO) standard that provides an authentication and authorization mechanism, with an interoperability between different security services in distributed environments. SSO invokes an ease of use access over the net for different web services. Three main objects are involved in SAML procedures these are: user, Service Provider (SP), and Federated Service (FS) [129].

SAML standard can be implemented in different scenarios, which depends on the business needs and limitations. However, all scenarios follow close similar procedures. In the mean time we focus on the most prominent scenario that initiates MDSBA framework in analytics. SAML is initiated by users, who log-in to the FS first, then request an access permission for SP. The access details and permissions are encapsulated in an XML file, known by Assertion. This file is transmitted by using SAML security tokens. Users may initially request access from the SP, if so, the SP redirects the users back to the FS authentication and authorization service. SP must redirect the users request to the FS or identity provider for authentication purposes. The request is formatted in assertion notations, which is a set of XML groups that are bind using several communication protocols such as: POST, GET, Representation State Transfer (REST), or Simple Object Access Protocol (SOAP) [130, 131]. Developers implement GET and POST on transferring very limited parameters, therefore, current SOAP and REST are widely used communication protocols in web development. The major difference between SOAP and REST is the performance and speed. REST implements JSON file to transmit parameters as a faster option than XML. However, REST performance clearly appears in mobile applications, while in desktop applications, SOAP and REST are quite similar in performance-wise [132].

MDSBA is interoperable with two variant binding methods, and these are; front-channel bind, and back-channel bind. The front-channel bind adopts the traditional transfer methods of GET and POST. It is triggered using the web browsers between users, in FS side, and servers, in SP side. This conventional method of web communication implements the standard security of TLS/SSL, and public and private keys over TCP/443 [133]. The second binding method is usually through server-server. Its notion applies two servers communicate with each other, and without human interaction, by initiating connection through a pre-defined port. The first server is located in the FS side, while the second server is located in SP side. Servers are essential to complete XML transmission through SOAP/REST servers. SOAP is used parallel with a set of artifacts. The communications procedures start by the FS artifact generator, which generates and transfers a source ID, references, and messages. This artifact is transferred by using the front-channel method or back-channel method, as described before. The SP verifies and recognizes the received artifact messages. The SP generates an XML request, known as <ArtifactResolve>. This request is transferred back to the FS. The FS generates an XML <response> or assertion and sends it again to the SP. Figure 5.1 summarizes SAML communication steps between SP and FS [134].

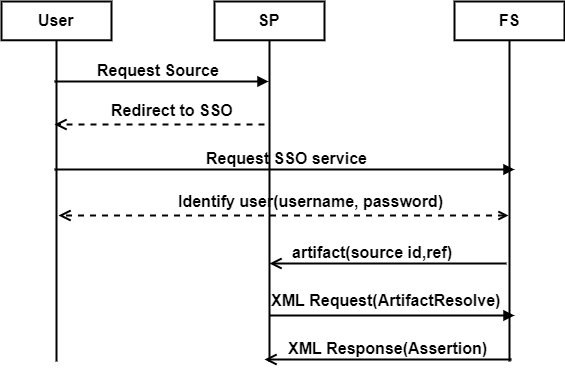


Figure ‎5.1- SAML communication steps between SP and FS

The FS or (idP) builds a SAML XML response that contains several sections of assertions. The HTML form encrypts the XML assertions on transmitting to the SP. The response can be sent using POST, REST, or SOAP. SAML is an XML-based single sign on (sso) standard, which provides authentication and authorization mechanism, with an interoperability between different security services in distributed environments [129]. SAML standard can be implemented in different scenarios, which depends on the business needs and limitations. However, all scenarios follow close similar procedures as in [130, 131]. SAML 2.0 assertions divide the XML file into 8 sections as follows: response ID, Issuer ID, Status (success or Fail), Assertion ID, Signature key, Conditions, Authentication statement, and Attribute Statement. The last section contains unlimited names and values of attributes. Developers use this section to pass any authorization attributes and values [133]. In MDSBA, we need to inform the access control system with the ownership and access levels, so MDSBA algorithms calculate the sensitivity values for each permitted Q-ID group [134]. This can be embedded in the Attribute section in the XML response file. The assertion can be formatted similar to this example in Figure 5.2.

Figure 5.2 presents the XML response in SAML assertion sections. This response is transmitted from the idP or FS to the SP. MDSBA transmits some parameters through the attributes section. Other parameters are available in the SP side with XML format, and there is no need to transmit them on every user’s access. The other parameters describe the dataset details of Q-ID attributes, general attributes, class values, *k-anonymity* value for each data attribute, taxonomy trees, Q-ID probabilities, and others. The assertion XML file does not contain all needed parameters for MDSBA calculations. It only contains various parameters that should be updated as per user’s access level. Hence, variable like, ψ, and data status are mutable and must be given to every separate user’s access.

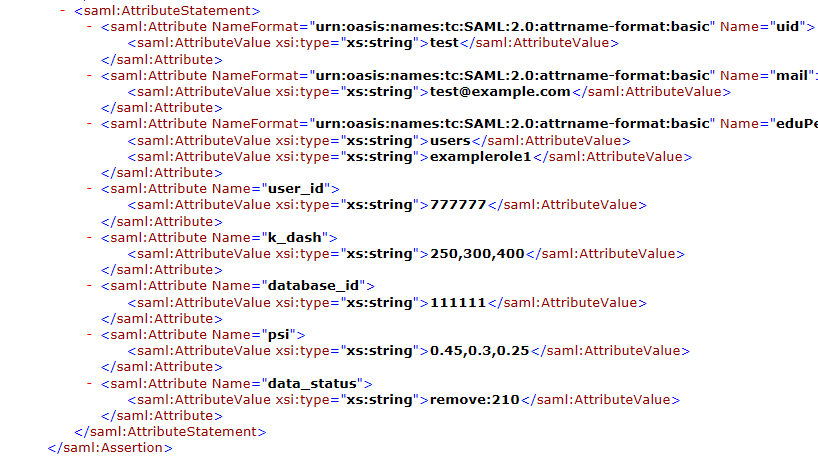


Figure ‎5.2- XML response in SAML assertion sections, and attributes section modification

## MDSBA and Granular Access Control

MDSBA was implemented for data analytics operating in parallel distributed operations. Hence, the gradual security anonymisation should be considered in manifesting the large number of big data accesses. Since multi-user access requires fine-grained access on the data level, MDSBA provides a granular access by implementing gradual levels of anonymisation. As described in chapter 3, data attributes are divided into groups of two to four Q-ID attributes. Anonymisation is applied separately to each group. Data owner can decide the level of data suppression or masking on each group. For instance, Doctors do not need to know operation’s costs, and financial status for patients, hence, data owner prefers to highly anonymise financial status data and lightly anonymise health status data or group.

MDSBA aims to fragment data nominally to reduce the data size on accessing the anonymisation software program. This can be implemented by bundling a number of Q-IDs and Classes into small groups horizontally. Medical information may contain more than one class attribute, and a considerable number of Q-IDs. Eventually, the increased number of Q-IDs may produce a massive computation cost and data overflow, which may unexpectedly terminate the anonymisation program. Hence, dividing the Q-IDs into small bundles horizontally can support the performance and scalability. However, the Q-IDs are determined by data owners and can be divided randomly or logically. Dividing Q-IDs logically supports the granular access control and reduces the anonymisation loss. Q-ID grouping can be divided based on user’s roles, where user’s privileges and interests varies from one role to another. User’s roles refer to organization’s departments, and each department conducts certain analytics to abstract the information of interest. For instance, HR department may need to know the personal user’s information, while financial department focuses on the user’s financial status. Giving the right amount and value of information should be controlled to alleviate information overload and improve the security level.

### Live Data and Archived Data

Data owner needs to assign Q-IDs and Q-ID groups, and give a proper *k-anonymity* value for each group. Choosing the optimal *k-anonymity* value will be discussed in the next chapter. Data owner, also needs to map Q-ID groups to user’s roles as explained before. In data lifecycle management, non-used data needs to be archived to improve the storage capacity, and to provide the non-active data with a long-term retention. Eventually, data can be categorised in to two types of live and archived data. Data owner needs to determine the type of each data (live, archive). That is if data is live, then the previously mentioned aging factor in Equation 3.7 can be ignored, and the Equation 3.6 will be ψ=ω. The reason for omitting the aging factor term is data continuous update, hence, live data importance will not be degraded with the time factor. Data analytics consumes an intensive computation time, which makes the real-time analytics hard to obtain. In this case, live data can be anonymised with an acceptable time that is like a real-time. This is essential to obtain a better accuracy in analytics results. Productive database actively varies every time, and the growth of data depends on the data nature. The anonymised copy of live or productive database may become old in few days, or even hours. This depends on the changing and growth rate of data.

In MDSBA framework, a practical solution was added for archived data. This solution anonymises data for every specific user only once. The user uses the same anonymised data on every access, and there is no need to anonymise the same data each time. This anonymised data can be used for a long period of time, before getting a replacement with a new anonymised copy. This solution saves the user’s time, and improves all over performance. However, this solution cannot be implemented in live data, which is updated regularly. Hence, live data was controlled by a (life-time) parameter. The parameter is determined by data owners. The anonymised copy of data remains during a life-time period. When life-time period expiries, then the anonymised copy is prone for removal. When the user accesses the same dataset, the anonymised copy is verified by life-time expiry, if so, then the database copy is purged, and a new anonymised copy will be generated. For example, if the life time was determined by the data owner as 24 hours, then the anonymised copy of data will be erased, if the user attempts to reuse data after 24 hours, and a new anonymisation process will take a place to create a new anonymised copy.

Figure 5.3 and 5.4 illustrate the steps that data owners need to follow on creating and configuring a new dataset. The figures show the distinction options between archived and life data that data owners need to provide. The live data requires the expiry number of hours before it is being purged, as shown in Figure 5.3. The archived data requires more parameters as shown in Figure 5.4.

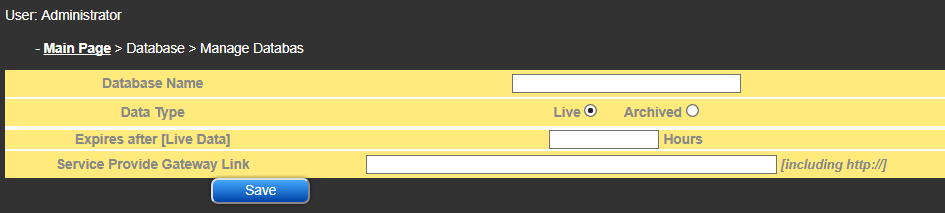


Figure ‎5.3-Live data options for data owner interface

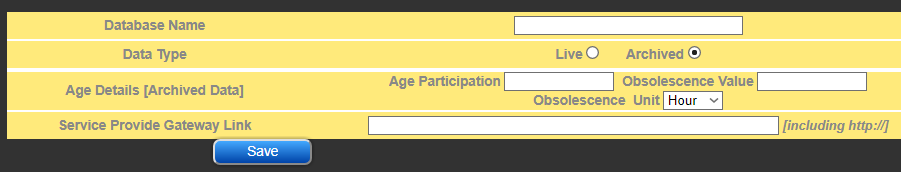


Figure ‎5.4-Archived data options for data owner interface

Figure 5.5 summarises the steps of preparing any new dataset for analytics. Data owners need to define the Q-ID attributes, assign them in groups, assign a *k-anonymity* value, and map them to business roles. Each Q-ID attribute must be defined by a probability value, and a masking type. Dataset is categorized as Archived or Live data, where the archived data is given the time factor parameters of; age participation, obsolescence value, and obsolescence unit. Apart from dataset preparation, data owners need to create business roles, users, organizations, and assign a set of roles to each organization. Users are created and managed by the delegated organizations.



Figure ‎5.5- Steps to configure initial values for each dataset

### MDSBA *k̄* percentage and Business roles

One of the granularity structure, in MDSBA, is assigning business roles to Q-ID groups. This is essential to formalize the access control as per business roles. Data owners prefer dealing with permissions using roles, rather than dealing with Q-ID groups. Moreover, business roles can be given variable parameters depending on user’s permissions, while Q-ID groups keep their anonymity parameters regardless the user’s permissions. This imposes two separate *k-anonymity* parameters; *k* value and *k̄* value. The value of *k* is assigned to Q-ID groups, as unchangeable fixed value to represent the optimal *k-anonymity* value from the grouping prospective. In contrast, the value of *k̄* is assigned to business roles, to represent the user’s access permissions. Therefore, *k̄* value is assigned to business roles that are delegated to organizations. Assigning value imposes knowing organization’s needs of business roles. One more parameter was derived from *k* and *k̄* values relations, known as *k* percentage (. This value is defined as the permitted level that allows a user to view data. For instance, the percentage of =100% implies that the user is given. The value of =60% implies that the user is given 60% of the *k* value, so if the *k*=100, then. Organizations are given the business role along with the value. The *k̄* is calculated as *k̄* =*.*

The anonymisation process needs two parameters to apply the proper amount of masking, and these are *k* and *k̄* values. These two values are essential for calculating the value of ψ. The data owners set up each organization details, by determining the for each role. The is decided based on the service level of agreement between the data owners and the organizations. Let us study this example, which shows the sequential steps of preparing the Q-ID groups, mapping them to business roles, and assigning them to users. Table 5.1 shows a set of attributes for (Properties) data consists of {suburb, street\_name, property\_type, bedrooms\_num, bathrooms\_num, sale\_price, cost\_price, max\_offer\_price}. Such attributes can be divided into two Q-ID groups, so the first Q-ID group can be G(QID)1= {suburb, property\_type, bedrooms\_num, bathrooms\_num} with one class of {street\_name}, while the second Q-ID group can be G(QID)2= {sale\_price, cost\_price} with one class of {max\_offer\_price}. In this example, the data owner may create a set of roles for G(QID)1 = {Strata Manager, Sales Representative}, and for G(QID)2= {Construction Contractor, Strata Manager}. The given roles determine the data of interest for each role. However, users can be given more than one role, and each role can be mapped to many Q-ID groups, as shown in Figure 5.6.

Let us refer to the previous property example, as shown in Table 5.1, by considering a user U1 belongs to ABC Company. The company was given three roles of {Strata Manager, Sales Representative, and Construction Contractor}. The company was given three roles to delegate users any role of the three given roles. The user U1 was given two out of three roles. The owner assigned *k*1=500 for the G(QID)1 group, and *k*2=700 for the G(QID)2. Moreover, [Strata Manager]=0.7, and [Sales Representative]=0.8 for ABC company. The permission of user U1 will be given based on the percentage of. Hence, the first *k̄* value for the first group is *k̄* = 0.8 × 500 = 400. In the second Q-ID Group, Strata Manager is the only given role that is mapped to G1, so *k̄* = 0.7 ×700 = 490. The final result shows the user U1 with the following *k* and *k̄* ordered values U1(*k*, *k̄*)={(500,400),(700,490)}.

Table ‎5.1- Illustration example of Properties data for roles and Q–ID groups

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **G(QID)1** | **G(QID)2** | ***k*1** | ***k*2** | ***k̄* perc for Roles [ABC company]** |
| Suburb | cost\_price | 500 | 700 | Strata Manager (70%)  [Mapped to G(QID)1 & G2(QID)2] |
| property\_type | sale\_price |  |  | Sales Representative (80%)  [Mapped to G(QID)1] |
| bedrooms\_num |  |  |  | Construction Contractor (10%)  [Mapped to G(QID)2] |
| bathrooms\_num |  |  |  |  |
| **CLASS:**  street\_name | **CLASS:**  max\_offer\_price |  |  |  |

In the previous example, since there are two Q-ID groups available in this dataset, then two-pair of integers will be created to be transmitted to the SP by SAML. The number of created integers must be equal to the number of Q-ID groups. If the number of Q-ID groups are three, then three-pair will be transmitted and so on. The pair consists of values for each Q-ID group. The pair represent the permission decision for each group, that is, if, then the Q-ID group is suppressed, else it is permitted. The SP-Gateway interprets this notation accordingly. For example, let a user U2 is given the following; U2()={(80,80),(100,80)}. This notation is interpreted on the SP side as; G1[suppression], and G2[permission]. In another example, if a user U3 was assigned to the three available roles, then the two-pair for G(QID) are given as: *k̄* for Strata Manager is *k̄* = 490, for Sales Rep. is *k̄* = 400, and for Construction Contractor is *k̄* = 630. G(QID)1 is given the lowest *k̄* value, which is *k̄*= 400. Similarly, the G(QID)2 is given the lowest *k̄* value, which is *k̄*= 490. The final result is transmitted to the SP as follow: U3()={(500,400),(700,490)}, because Strata Manager role is mapped to both of the Q-ID groups.

The organizations are delegated to control their own user’s permissions and roles assignment. However, organizations cannot exceed the pre-determined given values by the data owner. Eventually, users inherit these percentages from their own organizations. Also, each organization is given limited roles with for each role. Figure 5.6 illustrates the G(QID) to role mapping with the organizations intersections.

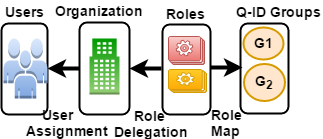


Figure ‎5.6- Organization-Roles-Groups mapping

### MDSBA Three Services

MDSBA framework starts from the Federation Service processes and ends with the user’s analytics processes. The framework consists of three main services; core, initialiser, and anonymiser. The core resides in the Federation Service side, while both of initialiser and anonymiser reside in the Service Provider side as shown in Figure 5.7. MDBSA framework consists of four groups of servers these are: FS, SP-Gateway, Kerberos and LDAP, and Hadoop domain. Customer is defined as any user who attempts to access Hadoop domain for data analytics, and can be the data owner or a customer from any external organization. However, the external customer must be approved by the data owner. Both of FS and SP-Gateway operate SAML v2 servers for data transmission between SP and FS. Also, SP-Gateway contains the initialiser service, which remotely creates the anonymisation script. The anonymisation script is executed in NameNode servers. Hadoop domain is managed by a domain server, which consists of LDAP and Kerberos services to provide security and user’s authentication and authorization. Users are authenticated twice with two separate authentication accounts. The initial authentication is in the core service, while the second authentication is especially for Hadoop domain and located in anonymiser service.

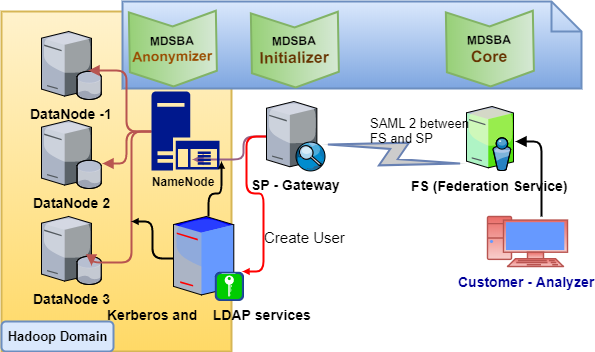


Figure ‎5.7- MDSBA three main services, core, initialiser and anonymiser

Figure 5.8 illustrates the sequence diagram for MDSBA three services; core, initialiser and anonymiser. The user requests an access to Hadoop domain for analytics tasks. The FS authenticates the user and determines the belonging organization and roles attached to it. The service, also, determines the user’s own roles assigned by his/her own organization. The generated XML request by SAML contains attributes with information about user id, data id, data status and the other sensitivity parameters. SAML, thereafter, takes over the communication procedures between FS and SP. The XML assertion, then is transferred to the SP-Gateway, which in turn receives the request, and retrieves the data information with the help of the data id that was received from the assertion XML. The SP-Gateway contains a set of files as described in Table 5.2. One of the essential XML files is definder.xml, which contains the necessary information about the requested data. The Core service is competed when SAML takes over.



Figure ‎5.8- Sequence Diagram for MDSBA framework.

As shown in Figure 5.8, the algorithm illustrates both services of initialiser and anonymiser. The Initialiser service starts by creating the Pig script, which is generated by collecting information from parsing two XML files, definder.xml, and SAML assertion file. The Initialiser service verifies the user’s availability from the LDAP / Kerberos server. The FS provides a user id, which is considered as the username on the SP side. Hence, the username that is generated from the FS user id, will be created in the LDAP server, if it wasn’t created before. In the third service data is copied to Hadoop domain, and specifically to the NameNode. If the username was created before, then the anonymiser service verifies whether to purge the previously anonymised data or keep. If data was purged, then the Pig script is triggered and a new anonymisation process starts. Later the user is provided with all access details including username, password, and data location of input and output paths.

#### Core service

The core provides the initial authentication and authorization for multi-access user’s level. The core service consists of a data schema and user interfaces to update MDSBA users and datasets details. User’s details include: user name, login details, organization’s names, general roles, and roles assigned to the organization. Data owner configures initial settings for each targeted dataset that users wish to access and analyse. The core service initial setup are shown in Figure 5.5. The aim of the setup is identifying the Q-IDs and classes in the data schema. Each two to four Q-IDs and one class are assigned to one group. The chosen Q-IDs group and a class are not necessarily found in one table. The created roles conventional names should mimic the user’s actual roles.

Figure 5.9 illustrates the Entity Relationship Diagram (ERD) for MDSBA core schema. The schema is located in FS side. The relationship diagram describes a basic information data diagram to store organization’s users, and roles. The diagram presents a (database) entity, which contains the {data\_name, creationDate, liveObject, ageParticipation, obsol\_VALUE, obsol\_UNIT}. The interface of this entity is shown in Figures 5.3 and 5.4. The attribute of (liveObject) presents the expiry time in hours for the live data. Hence, if data type is archived, then liveObject value is zero. If data type is live, then ageParticipation, obsol\_VALUE, and obsol\_UNIT are set to zero. As shown in Figure 5.9 ERD, the data owner is responsible about initialising the objects of: database, assign, organization, role, and qid\_group. The entity (role\_qid) is an associative entity to resolve the many-to-many relationship. The (assign) entity is also another associative, which contains an essential attribute of *k* percentage. Other entities are delegated to organizations to manage their own user’s details including; name, authentication, and roles assigned to users. Actually, business roles are given to the organizations, and not directly the users. Therefore, users do not have the privileges to obtain any business role beyond his/her own organization’s roles.

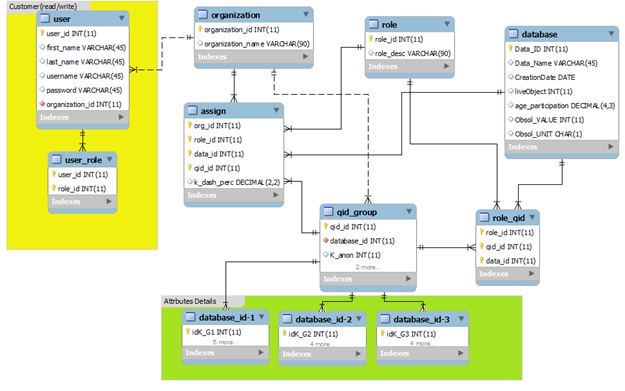


Figure ‎5.9-The core part of MDSBA framework

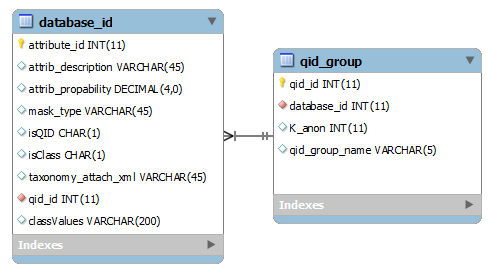


Figure ‎5.10-The core part of MDSBA framework

Figure 5.10 presents the data owner grouping for Q-ID attributes. The line shows a one–to-many relationship between both tables. Each created dataset imposes an auto creation for a new table named as database\_id. The new table lists all the dataset attributes, with full details of: description, probability, and masking type (taxonomy, integer, or suppression). The attributes (isClass) and (isQID) are Boolean values. Any data attribute can be either a class, Q-ID, or an ordinary attribute. Hence, three possible value-pair are assigned to both attributes; (isQID, isClass}={(0,0), (0,1), or (1,0)}. If both attribute’s values are (0,0), then the attribute is an ordinary attribute. If both attribute’s values are (0,1), then the attribute is a Class. Figure 5.10 describes this relationship between qid\_group, and the database\_id entity. One of the database\_id attributes is classValues. This attribute lists all available values, if the attribute was chosen as a class. These values are essential during the late anonymisation stages. Figure 5.11 shows one of the administrator’s interfaces that is able to create Q-ID groups. The main parameters are: choosing the Q-ID attributes, the *k* value, the class attribute, and the classValues.

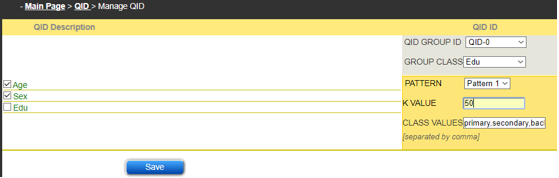


Figure ‎5.11-Interface to create G groups of Q-IDs, classes, and classValues

##### Keep or Remove Decision

Core service generates several XML files for transmitting them to the service provider side. The services include assertion.xml, definder.xml, and taxonomy trees files. The assertion file is generated by SAML server. The generation process requires some parameters read from MDSBA data schema. The service provider needs to know whether to keep the anonymised copy of the dataset or to remove it. The SP offers a service to purge the anonymised copies on a regular period. This service needs to know the exact expiry time for the anonymised copies. To do so, the core service examines two statuses of data types, when users attempt to access the service provider datasets. Live (production) or archived data are distinguished before determining the expiry time of the anonymised copies. Both cases are related to figure 5.3 and 5.4. The live data is related to the expiry period setup by the data owner, while the archived data is related to the obsolescence value setup by the data owner as well. The core service generates the assertion.xml file as per this available information.

The assertion xml file contains one attribute, known by data\_status. This attribute presents two values for the data status (keep or remove: time period). If it was the first time for a certain user to access the specified dataset, then initial value is presented by (remove:), where denotes the remaining time period calculated by the FS side. This value is interpreted in the SP side as: a new anonymisation process takes a place and the anonymised copy will last for the time period provided. For instance, if the value provided was [remove:6], then a new anonymisation process will start, and the expiry time for this copy is 6 hours. If the same user has accessed the same dataset after 4 hours, for instance, then the status will be replaced by [keep]. Notice that the status [keep] does not need to be presented with the remaining time. If the user has accessed again on the next day, then the status will be replaced by [remove: 6]. The SP side will look at the status and store the time period added to the status [remove: remaining time].

##### Initial Access

Usually, the FS determines keep or remove, by investigating the latest user’s access for the requested dataset. In the initial access, when the user has never accessed that specified dataset yet, then an initial value is created. Let us formalize the expiry time for each dataset. Suppose that the current date/time is denoted by, and the expiry period given by the data owner is denoted by. The remaining time period for the specified dataset *D* and user *U* is denoted by. For the user’s initial access when data type=live, the equation used in FS side is:

(1)

For the user’s initial access when data type=archived, the obsolescence value Ø is replaced by the expiry period. The equation depends on the date and time that the dataset D was initially uploaded, which is denoted by. Thus, the equation used is:

(2)

In the archived data, the value of falls between two-time intervals: the first interval when the user accesses the dataset before the end of the obsolescence period. The second interval when the user accesses the dataset at or after the end of the obsolescence period. The first interval imposes that (, which means that the user’s access was before the end of the obsolescence value. The second-time interval imposes that (, which means that the user’s access was at or after the end of the obsolescence value. Based on these two intervals, the keep/remove statuses are determined. The initial access is determined if the access was before or after the obsolescence value. If the access was before, then the status will be [remove:], as illustrated in equation 2. If the access was equal to or after, then the status will be [remove:]. The time period of *E(T)* can be determined by the data owner, for instance, one month or so. On the other side, the SP interprets this value differently. The SP stores value in the log file to purge the created dataset after the expiry period. For instance, if the data owner assigned (=74), and the user’s access date and time was 5-Dec-2017 at 13:10. Initially, the FS will transmit the value of [remove: 74]. The SP will purge the anonymised data at the end of the expiry time, which is 5-Dec-17 13:10 + 74 = 8-Dec-17 at 15:10. The 74 hours conclude three days and two hours. Therefore, the dataset will not be available after 8-Dec-2017, 15:10. During the past three days, any access attempt will provide a data status of [keep], which allows users to use the previously anonymized copy.

##### Subordinate Access

After the initial access, the FS determines (keep or remove), by investigating the latest user’s access for the requested dataset. If it was found that the user has already accessed the specified dataset, then the expiry period value is calculated for data type=live as:

(3)

The FS refers to the latest access for the specified dataset to determine whether (keep/remove]. The above equation is used to help the decision making. The expiry is calculated, and compared with the expiry period that was assigned by data owners. If data type=archived and the access was before the end of the obsolescence value, then the status will be [keep]. If data type=archived and the access was equal to or after the end of the obsolescence, then the status value will be [keep], if the time period *E(T)* has not expired yet. If the time period *E(T)* has expired, then the status will [remove:].

#### Initialiser Service

This is the second component of MDSBA framework. The service operates in the SP side, and outside Hadoop domain. SP should provide a gateway service, which is considered to be the first entry level for users who wish to participate in data analytics. For security purposes, the gateway server should be isolated from Hadoop domain. This service provides initial documents to prepare data for anonymisation. SP needs to provide a valid path in the SP-Gateway server for data owners. Each uploaded data on SP servers must be provided with essential set of documents to support anonymisation process. Data owners need to upload these supportive files over a given path in the SP-Gateway. The files consist of two XML files, one ecosystem script such as pig or hive, and the anonymisation software program such as Java Jar. One of the XML files is the definer.xml, which defines Q-ID Groups and their related Q-IDs and the other attributes. The format of this file is described in Figure 5.12. The file contains database Id, path, attributes, and Q-ID groups, types and classes. These attributes are needed to provide complete information about data on the SP side. The Q-ID types are given the letters (i,s,t) to denote the interval, suppression, and taxonomy tree consequently.

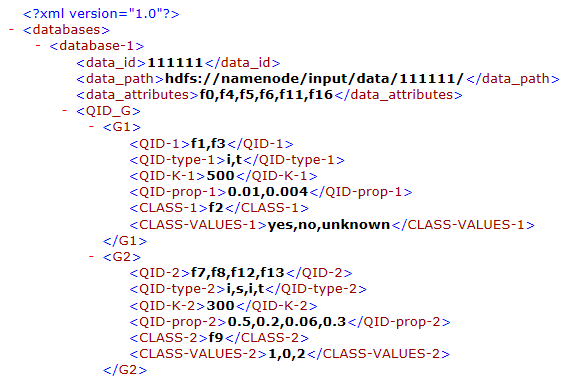


Figure ‎5.12- Definder.xml file demonstration

The file (definder.xml) was generated from the core service. All the information are stored in the data schema and can be easily generated as per dataset. The file consists of Q-ID groups created from data schema as shown in Figure 5.12. SP-Gateway server communicates with the FS throughout SAML v2 server. On user’s login, the core service generates another XML file of SAML assertion (assertion.xml), which contains login and anonymisation information in the attributes section of XML file, as shown in Figure 5.2. The XML assertion file is unlike the definer.xml, since it is generated on each time a user requesting access, while the definder.xml is transferred to the SP-Gateway only on dataset alteration. The attribute section of assertion.xml file contains the following information: user id, database id, *k̄* values, sensitivity level ψ, and data status. Data status consists of two different values; keep or remove. The data status defines the previous anonymised copy and its validity for the same user. This decision is made on the FS side, and based on the last login time by the user. The decision is made by comparing between the last login, and the obsolescence value for the archived data or the liveObject for the live data. Table 5.2 illustrates the essential available files for each dataset on the SP side, as a part of the initialiser service in MDSBA framework.

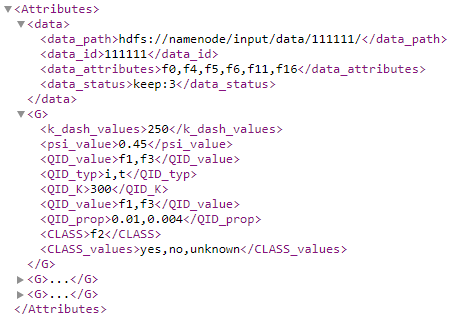


Figure ‎5.13- Sample of data\_id.xml file

Table 5.2 shows the output files of pig script, and data\_id.xml. The data\_id file is very important for Java programs during the anonymising process. Java files, such as ADJUST.java, SSG\_P1.java, SSG\_P2.java need some parameters to execute the program. The file data\_id.xml is a combination of definder.xml and assertion.xml files. The file is generated by the initialiser service, which contains the following attributes: All information contained in definder.xml + some information abstracted from the SAML assertion including (data\_id, user\_id, *k̄*, ψ, data\_status {keep, remove}.

Table ‎5.2- List of files used by the initialiser service

|  |  |
| --- | --- |
| **File name** | **File description** |
| **MAIN FILES** | |
| definder.xml | Generated by core service, and uploaded by data owner once, or on data alteration. It contains data information including:   * Attributes named by f sequence: * Q-IDs named by f sequence * Q-IDs anonymisation type { taxonomy: t, interval: I, suppression: s}, * K value for each Q-ID group * Q-IDs probabilities * Class named by f sequence * Class values |
| assertion.xml | Generated by the core service, which contains the following attributes:   * All information contained in definder.xml + some information abstracted from the SAML assertion including (data\_id, user\_id, *k̄* , ψ, data\_status {keep, remove} |
| **OTHER FILES** | |
| User\_id-i.pig | Set of pig scripts files. Generated by the initialiser service |
| Anonymiser.jar | Contains six java files: ADJUST.java, SSG\_P1.java, SSG\_P2.java, NG\_P.java, SUP\_P.java, OBV\_G |
| Taxonomy trees [fi.xml] | Generated by core service, and uploaded by data owner once, or on data alteration. Each file represents one Q-ID attribute of taxonomy tree type. This files is used by Java files during the anonymisation process. |
| Data\_id.xml | Generated by the Initialiser service, which contains the following attributes:  All information contained in definder.xml + some information abstracted from the SAML assertion including (data\_id, user\_id, *k*̄, ψ, data\_status {keep, remove} |

Figure 5.13 presents an example of data\_id.xml file. The example shows complete details about the dataset id, path, attributes, status, and details of each group. The data status is presented by keep: 3, which indicates the dataset can be kept for the next three hours. Similarly, if the status showed remove: 8, then the dataset will be purged immediately, and the new anonymised copy will continue valid for the next 8 hours. The unit used in the data status is hours. If data type is not live (archived), then data will not be purged while the copy age is smaller than the obsolescence value. Usually, the obsolescence value is much larger than live time period. Therefore, the data status in data\_id.xml file is possibly large, such as keep: 4320, which indicates that the obsolescence value is 6 months, and data will be purged after six months.

##### Generating Pig scripts in Initialiser Service

The Pig script is essential to anonymise data before permitting users access. The script is generated based on the definder.xml, and SAML assertion.xml files. These two files provide the essential parameters to create a set of Pig scripts files. The anonymisation script will be transmitted to Hadoop domain. The initialiser service creates one Pig script file for each Q-ID group. 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.

The initialiser service reads definder.xml file to obtain data information including data location, Q-ID attributes, *k* values, status, probabilities, and classes. The initialiser first determines the data status, whether it is in (keep) or (remove) status. If data is in keep status, then no any anonymisation occur and the user can login to his/her previous HDFS directories. Else, the old anonymised data will be purged, and a new anonymisation process will start. Second, the initialiser creates the scripts line by line, starting by registering the Jar file, assigning the load location, filtering, creating the user HDFS directories, grouping and anonymisation by UDF Java. Figure 5.14 illustrates an example for Pig script created by the initialiser service.

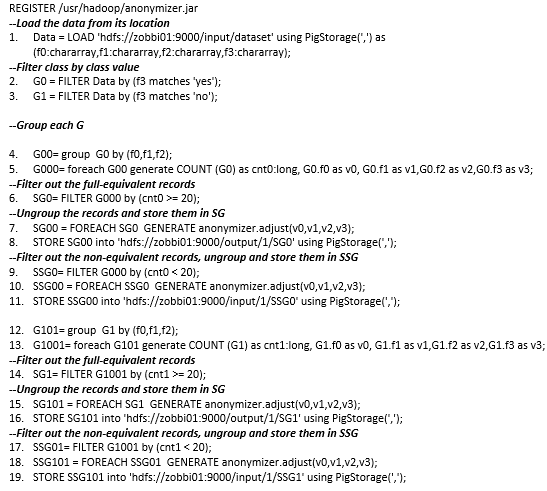


Figure ‎5.14- Created Pig script file by the initialiser service

Figure 5.14 describes part of the generated Pig script for three Q-IDs. The full script is attached in Appendix A. The script filters the class value as per two values of (yes, no). Hence, two G groups are created, and each group is aggregated by three Q-ID’s, then by two Q-IDs, and finally by one Q-ID. In Pig Latin, it is possible to name attributes with (f) letter instead of using the attribute name. Before generating the Pig script, the initialiser initiates two HDFS directory paths for the user. However, the directories are not created so far, but they should be named with the standard of input and output as follows:

*hdfs://namenode:9000/[data\_id][user\_id]/input,*

*And hdfs://namenode:9000/[data\_id][user\_id]/output*

Two HDFS directories are created for each user, and two Pig scripts files will be generated based on the given information. The HDFS path consists of the NameNode, followed by the port number, the data ID, and the user ID. The default port number is usually 9000 or 8020. This given path will be created in the anonymisation service stage, as it will be described in the next section.

#### Anonymiser Service

This service operates between the SP-gateway and the NameNode, and specifically in both of the gateway server and the NameNode server. The service process is located in SP gateway to copy and execute scripts in NameNode server remotely. The process of the service copies the output files from the initialiser service. The files include XML, Pig, and Jar files, as shown in Table 5.2. The anonymiser service executes anonymisation scripts in Hadoop domain. The domain is controlled and managed by LDAP and Kerberos server, which provides authentication and authorization services of domain user’s access. Also, Kerberos is enabled to support tickets granting for Hadoop services, which creates Hadoop secure mode.

The anonymiser service executes shell commands by implementing web programming. This technique is secure and possible with the new web programming extensions and libraries. MDSBA adopted PHP programming language, with the support of phpseclib, php-devel, php-pear, and libssh2-devel extension packages. The packages perform properly with the SSH2 [135]. SSH2 can access other devices to execute commands via an interactive shell with read and write permissions, as shown in Figure 5.15. The command script is contained in a secure web page to protect the secure domain sudoers account related to the executed shell commands. The web page can be either by a trusted certificate or even by a self-signed certificate. The used account should have full privileges over HDFS storage directory and part of LDAP sudoers [110]. There are subtle differences between LDAP sudoers and local sudoers, so the used account in the shown script must contain an LDAP domain account, as shown in bold [136]. The script executes the Pig script remotely, as shown in Figure 5.15 script. Before executing the script, the previously mentioned files, of scripts and XML files, must be copied to NameNode server. The command in SSH2 is given by *$command="scp /var/www/html/".$user\_name.$databas\_id."/\*.\* /$NameNode\_server";*

|  |
| --- |
| <?php  include('Net/SSH2.php');  $ssh=new Net\_SSH2('namenode.fullname');  $ssh->login('hadoop','password') or die ("Login failed");  ------------------------------------------------------------  $ssh->exec("hadoop fs -mkdir /[data\_id][user\_id]/input");  $ssh->exec("hadoop fs -mkdir /[data\_id][user\_id]/output");  $ssh->exec("hadoop chown -R user\_id hdfs://namenode:9000/[data\_id][user\_id]/input/");  $ssh->exec("hadoop chown -R user\_id hdfs://namenode:9000/[data\_id][user\_id]/output/");  $ssh->exec("pig –x mapreduce script-1.pig");  ?> |

Figure ‎5.15- PHP sample for anonymiser service

MDSBA framework maps the user\_id of the Federation Service to Hadoop domain through LDAP service. The same user\_id is created in LDAP domain, if it was not created before. The anonymiser service verifies the user\_id availability with the LDAP service. If the user\_id is not available, then create a username and a random complex password. For security reasons, the password can’t be mapped with the FS password. Hence, a new created password is required to authenticate the user on accessing Hadoop domain. So the user can access the NameNode though SSH. If the user\_id has already been created, then verify the anonymised data whether to delete it or keep it, so the user can use the same previous authentication details. For the first time access, a shell command line is triggered to create the input and output HDFS directories, and finally run the Pig script, as illustrated by Figure 5.15 in PHP programming.

Figure 5.16 illustrates the SP anonymisation algorithm. The procedures describe both initialiser and anonymiser services to permit users accessing anonymised copies of the dataset. The procedures are summarised by parsing data ID and user ID from assertion.xml file. The assertion file provides some needed information to build-up the Pig Latin script, which creates anonymised copies of datasets. The first conditional statement, in the algorithm, verifies the whether the username was created before or not. The process interacts with the LDAP server, and fetches the username by the user\_id, where username=user\_id. If the username was not found, then it will be created by using the user\_id as a username, and a randomly created password. Next, the HDFS directories should be created and given the permission. Creating the user’s directories requires the knowledge of user\_id and data\_id. The initialiser service needs to read two XML files, definder and assertion, to prepare the needed documents for the anonymiser service. The initialiser first creates three main files of; Pig script, data\_id.xml, and SP log file. The data\_id.xml and the Pig script are passed to the anonymiser service, which in turn, copies the essential files to a temporary file in the NameNode of Hadoop domain. Finally, the Pig script is executed so the anonymisation copy of dataset is ready at the completion of the anonymisation process.

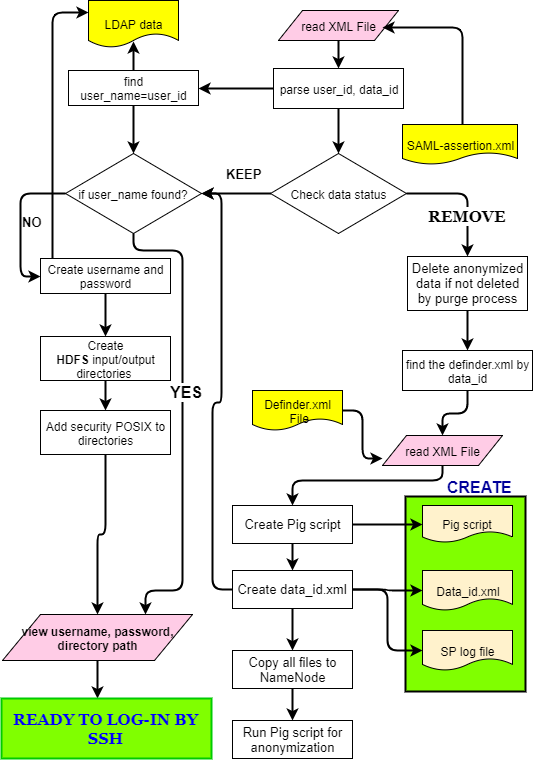


Figure ‎5.16- Initialiser and Anonymiser algorithms

|  |
| --- |
| Algorithm for Anonymiser service |
| 1. Login automatically by Hadoop username and password 2. Read the user\_id and check if it is available in the LDAP domain 3. If not found then create a new username=user\_id and a random password for the user 4. Create HDFS path for input and output by using Hadoop authentication 5. Give permissions to the username=user\_id for HDFS input/output folders 6. Create temporary folder in NameNode server 7. Copy all needed files to the temporary folder 8. Run the Pig Latin script 9. The user log-in to the anonymised copy by using SSH interface |
| Output: Anonymised copy of the original dataset |

Figure ‎5.17- Algorithm for Anonymiser service

Algorithm 5.17 shows two authentication levels by user, to access Hadoop domain. This is essential in securing Hadoop domain. The first authentication process is required by the FS to find out the user’s authorization level. The second authentication process is needed for SSH access. The password authentication is essential on accessing NameNode server through SSH interface. Users are able to use SSH to access Hadoop service at any time to submit analytics queries, and without going through the long process of the first sign-in. However, this shortcut may not provide users with the latest live dataset. The solution for this issue is creating a continuous process to purge the anonymised copy of the dataset. The purge process continuously reads the SP log file, in order to determine the purged data.

##### Service Provider Log File and Purge Process

The log file is a major component for any server service. It is a primary tool on tracking all activities occur on servers. For this reason, MDSBA creates its own service provider log files. One of the primary log files is created for a continuous process, known by purge process. The process mission is purging all expired anonymised datasets to utilize storage capacity. All types of datasets, live or archived, are affected by the time factor. The time is an essential parameter to determine the anonymisation and access level. MDSBA reduces the archived data masking, to allow more gained information, parallel with the time factor. Users should not use the same anonymised copies of datasets at all times. The current anonymised copies may not be valid after a certain period of time. Therefore, purging datasets and creating fresh anonymised copies is needed. The created log file must record the user ID, the data ID, the anonymised data path, and the expiry date/time, as shown in Table 5.3.

The FS side interprets the attribute data\_status of assertion.xml file. The attribute notation is presented by [keep/remove:]. The anonymiser service will not take any action when the status=keep, while few steps are considered if the status=remove. The remove status is always attached with expiry period of time, therefore, the anonymiser service registers part of the details in the log file, as shown in Table 5.3, and proceeds to the next following steps. The anonymiser, firstly, checks if the previous dataset was purged, or not. If it was not purged, then the anonymiser process will purge the previous anonymised copy. Secondly, a new anonymisation process will be initiated.

Table ‎5.3- Log file for the purge process

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **User ID** | **Data ID** | **Anonymisation Path** | **Expiry date/time** | **Last purge attempt** | **No. of Attempts** | **Purge Result** | **Purged (yes/no)** |
| 123456 | 111111 | */111111123456/input* | 05-01-18  15:20 | 05-01-18  18:10 | 4 | Fail | No |
| 321215 | 111111 | */111111*321215*/input* | 12-03-18  11:05 | 12-03-18  13:30 | 2 | success | Yes |
| 123456 | 222222 | */*222222123456*/input* | 25-03-18  01:20 | 24-03-18  03:30 | 3 |  | No |

Table 5.3 receives entries from two separate services. The first four attributes are added or updated by the anonymiser service. The expiry date/time is calculated in the core service. The anonymiser parse the data\_status attribute value, converts the value to a real date and time, by adding the current time to. The anonymiser stores only one record for each user attempting to access a specified dataset. As shown in Table 5.3, the user ID=123456 has only one record for accessing data ID=111111. Another record for the same user can be created, if the user accessed different dataset. Therefore, each record will be created once and updated on every user’s access.

The purge process cannot create any record, instead it just updates the available data records. The last four attributes, in Table 5.3, are updated by the purge process. The purge process is an infinitive process, which is considered to be a part of the anonymiser service. The process is triggered as per time schedule. It can be setup for an execution as desired, for instance every 3 hours. The process reads all records of the log file, where {(purged=null/No), (Purge Result ≠ Fail), and (expiry date/time ≥ current date/time)}. The purge process deletes all expired copies, by following the anonymisation path for each copy. After the deletion completion, the process updates the data record by {last purge attempt, No. of Attempts, purge results and purged (Yes/No)}. However, some copies of datasets may be busy with other analytics operations or queries at the time of triggering the purge process. In this case, the purge process will wait for a period of time and skip few purge rounds before trying another attempt. For this reason, the log file keeps the latest date/time for purge attempt. This delay is essential to give enough time for the query process to complete the task. The queries triggered by users may lock the datasets, while the process is ongoing. Therefore, trying to purge this busy dataset on each round is inefficient. Purge process uses some parameters stored in a setup XML file, known as purge.xml. The XML file is shown in Figure 5.18.

The XML file allows system administrators to setup the waiting period before the next round starts. Figure 5.18 shows the default value of waiting period, which is 30 minutes, while the next attempt is given by 3 times. This means that the purge process will try to delete the datasets, if failed, the next attempt will be commenced after 2 hours of creating the anonymised dataset. That is 3 attempts within 30 minutes waiting period between each attempt. Moreover, the process will try to delete the dataset 4 times, as configure in Figure 5.18. With each time, the log file is updated by incrementing the No. of Attempts, and updating the rest of the attributes. The attribute (Purge Result) is updated with the values success/fail after the fourth attempt. If the result value was fail, then an email will be sent to the system administrator’s email (admin@service\_provider.com), as shown in Figure 5.18.

It was explained that two different processes can delete the expired datasets. Both processes are part of the anonymiser service, and they read from the same log file. The anonymiser may delete datasets, only if data status is remove. However, the purge process is the main process that carries over the deletion operations.



Figure ‎5.18- Sample of purge.xml configuration file

## Improvement to MDSBA Security

Sweeney has described few attacks against *k-anonymity* method, as discussed in Chapter 2/ section 2.3.5. However, MDSBA maybe prone for different attacks. The previously mentioned attacks are not possible in MDSBA. This refers to the consistency of anonymity. In MDSBA, Q-IDs anonymisation refers to the attribute probabilities. The anonymisation always starts with the lowest probability attribute. Hence, amending queries does not switch the anonymisation process to other Q-ID attributes. Also, the class attribute, in MDSBA, must gain the *k-anonymity* equivalency principle. However, other possible attacks against MDSBA can be summarised into two types of attacks; obvious guess and Across Groups Unique Identifiers (AGUI). These two attacks are described in the next two sub-sections.

### Obvious Guess

An adversary may be able to guess the sensitive attribute (class), if the Q-ID attributes are known to the adversary. This violation may appear when a group of equivalent records have one class value. In this case, increasing or decreasing the number of equivalent records does not affect the security level. For instance, if a group of patients have one value of the class ‘Diabetes=positive’ and they share the same race, age, and state, then the intruder can obviously guess the diabetes state of the patient. This attack is simply defined as:

**Definition 5.1***: Obvious Guess attack may appear in MDSBA if a group of fully equivalent records consist of one class value.*

In the Obvious Guess, an adversary may easily guess the sensitive attribute (class), and without having to identify the record. For example, in Seer data, we may have a data bag shown in Table 5.4. In the described bag example, the diagnostic may contain one value only. This class demonstrates the Obvious Guess breach, which can be interpreted as; any black person lives in the county 125, and was diagnosed on 1997, with an age of 25 must have a positive histology cancer. The Obvious Guess occurs if the bag contains only one class value.

Table ‎5.4- Obvious Guess example

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Y. DIAGNOSIS** | **COUNTY** | **RACE** | **AGE** | **DIAGNOSTIC** |
| 1997 | 125 | Black | 25 | Positive histology |
| 1997 | 125 | Black | 25 | Positive histology |
| 1997 | 125 | Black | 25 | Positive histology |

### Across Groups Unique Identifiers (AGUI)

The second possible attack against MDSBA is caused by splitting Q-ID attributes into small groups. This unique identifier’s security threat is lower than the uniqueness appearing in each Q-ID group. The attacker needs to know almost everything about the victim’s background. This is believed to be a background knowledge attack, and defined as:

**Definition 5.2**: *Across Groups Unique Identifiers (AGUI) is a unique record that appears post the anonymisation. The record is not anonymised due to its equivalency with the other records. Hence, its uniqueness appears across multiple Q-ID groups and disappears on individual Q-ID groups.*

Based on this definition, we may notice that AGUI attack is possible if the adversary knows most of the personal attributes in multiple Q-ID groups. In this case, AGUI severity will not be as high as personal re-identification records that are targeted by *k-anonymity*. However, this low-security threat should be reduced to the minimal, to inhibit its impact on security violation. MDSBA does not guarantee AGUI prevention, but it supports the minimal appearance of such records. Dividing attributes into small Q-ID groups is essential to reduce the anonymisation computation cost, and to participate in the granular access process.

Table ‎5.5- Part of Seer data with two Q-ID groups

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Q-ID1** | | | | **Class-1** |  | **Q-ID2** | | | **Class-2** |
| **Age** | **Sex** | **County** | **State-county** | | **Year of diagnosis** | **Race** | **Grade** | **Diagnostic Confirmation** |
| 26 | Female | 31 | CT: Hartford | | 1991 | White | Unknown | Unknown |
| 26 | Female | 31 | NM: Chaves | | 1991 | White | Unknown | Unknown |
| 26 | Female | 31 | NM: Chaves | | 1991 | White | Unknown | Direct visualization |
| 26 | Female | 31 | WA: Jefferson | | 1991 | White | Unknown | Direct visualization |
| **26** | **Female** | **31** | **WA: Jefferson** | | **1991** | **Black** | **Moderately; Grade II** | **Positive histology** |
| 28 | Female | 31 | IA: Dubuque | | 1991 | Black | Moderately; Grade II | Positive histology |
| 28 | Female | 31 | UT: San Juan | | 1991 | Black | Moderately; Grade II | Clinical diagnosis |
| 28 | Female | 31 | UT: Utah | | 1991 | Black | Moderately; Grade II | Positive histology |
| 28 | Female | 31 | UT: Salt Lake | | 1991 | Black | B-cell; pre-B; B-precur. | Clinical Diagnosis |
| 28 | Female | 31 | WA: Jefferson | | 1991 | Black | B-cell; pre-B; B-precur. | Clinical Diagnosis o |

AGUI is caused by G(Q-ID) grouping, which may create another security concern. This may appear on dividing data attributes into small groups of Q-IDs and classes. Two to four Q-ID attributes are divided into groups with one class for each group. This grouping is essential to reduce the anonymisation computation cost, and to participate in the granular access process. However, this grouping may help adversaries to spot out few known records. To explain this, let us take a look at Table 5.5 that shows a small part of the Seer data, with two Q-ID groups.

### Resolving Obvious Guess

To avoid Obvious Guess breach, initial filtration on stage zero can be implemented, and before splitting class values into groups. For the sake of performance, and to avoid the data overflow on Java Heap memory, this kind of filtration was implemented by using a simple UDF program. The program only checks the equivalent records of data bag that is ≥ *k̄*, if only one class value was found, then the data bag will be transferred to SSG group. This group aggregates the attributes based on the highest Q-ID probabilities. The lowest Q-ID probability will not be aggregated. In another word, the Obvious Guess data bag is considered as semi-equivalent records, even with the fully-equivalent data records. Single class value is prone to data leakage and can be noticed before anonymisation starts. Figure 5.19 illustrates the zero-stage procedure, which protects data bags from Obvious Guess records.

The UDF program reads the fully-equivalent data records that are greater than, or equal to *k̄.* The program loops only once for verifying the class values. If one value found, then the data bag is stored in SSG groups. This algorithm implements a small number of arrays, with one iteration process, which enhances the all over performance and increases the process scalability.

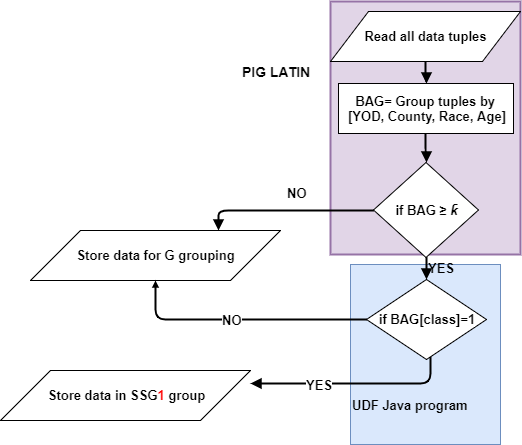


Figure ‎5.19- Stage zero of anonymisation to protect from Obvious Guess

### Resolving Across Groups Unique Idntifier (AGUI)

Resolving AGUI issue can be implemented by increasing the value of *k* for each Q-ID group. Therefore, assigning a quite large *k* value is highly recommended in MDSBA. The large *k* value leaves enough range of numbers to be assigned to the ownership levels, which provides the granular access method. Also, the large *k* value increases the anonymisation percentage dramatically, which makes AGUI appears less frequent. Also, the large *k* value decreases the appearance of obvious guess records. It was experimentally proven, that obvious guess records may gradually decrease parallel with the *k* value increase, while data anonymity dramatically increases parallel with the *k* value increase. However, the continuous increase of *k* value may negatively affect the information gain. Therefore, a trade-off between anonymisation loss and security threats reduction should be considered. The value of *k* should be moderate, which means neither too large nor too small.

Let us consider that Table 5.5 data has been anonymised with *k*(QID1) = 5, and *k*(QID2) =4. Each Q-ID group is anonymised separately. The highlighted record can be easily identified by adversaries, since the first five equivalent records of Q-ID1 overlap with the second four equivalent records of Q-ID2 [17]. Generally, the increase number of Q-IDs in any data may expose data to a higher rate of attack. This is because of the increase number of personal identifications or attributes. For instance, if knowing the patient’s age, gender, and postcode, may lead to uniquely identifying the patient with a probability of 87% [137], then the increase number of identifiers would increase the patient identifying even higher. However, Table 5.5 scenario possibility is low, because each Q-ID is anonymised individually, which allows a chance of randomness between both Q-ID groups. Regardless the scenario occurrence probability, there should be distinctive procedures to prevent or minimize the possibility of such occurrence.

## Experimenting Data Disruption in MDSBA Framework

In this chapter, MDSBA core structure was examined and tested with the defined services, and the security threats and resolutions. To evaluate MDSBA framework efficiency, a small lab was designed to demonstrate the three MDSBA services and the impact of security threats on them. The lab was setup at the University of Western Sydney as shown in Figure 5.7. The aim was to do an experiment about the framework components of the core, initialiser and anonymiser services. The lab comprises five virtual machines (VM) and one laptop. The five VMs are divided by one NameNode and two DataNodes for Hadoop domain. The forth VM was setup as an SP-Gateway, and the fifth VM was setup as an LDAP/Kerberos server. The laptop demonstrated the FS server, with a software application programmed in PHP language, with MySQL 5.6 database management system. Also, SAML server (Gluu Server [138]) was setup in both sides of FS laptop and SP-Gateway. SAML was successfully embedded in the user’s attributes in the XML assertion file, while transmitting attributes between both ends.

The experiment has adopted Seer dataset, with two G(QID) as shown in Table 5.5. An Organization named A was created and delegated to three different roles. The created roles are; HR Officer with = 40% (mapped to Q-ID1), Oncologist with = 40% (mapped to Q-ID2), and Hospital Manager with = 60% (mapped to both Q-ID groups). The organization and roles delegation percentages are illustrated in Figure 5.20. Also, three users were created with the following roles; user1 (HR Officer), user2 (Oncologist), and user3 (Hospital Manager).

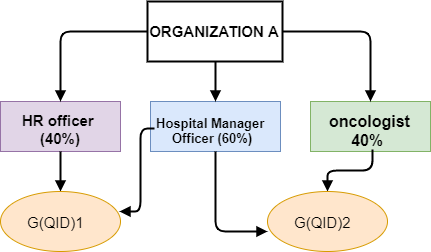


Figure ‎5.20- Organization A and delegated roles

The granular principle promotes a trade-off between the level of anonymised data and information gained. Fragmenting Q-IDs into groups to enhance the access control method is accepted technique if it does not contaminate the information gained predominantly. On the other hand, Q-ID groups may cause a security violation. The violation may occur with the multi-Q-ID access, while the anonymisation is conducted on each Q-ID group separately. Some non-anonymised records are sufficient enough to create AGUI. These identifiers may occur as a result of anonymisation randomness, which is conducted for each Q-ID group independently. The AGUI records appear when a user is permitted to access more than one Q-ID group. However, accessing more accessible Q-ID groups does not necessarily increase the number of AGUI records appearances. To reduce the AGUI in multiple Q-ID groups, an experiment should investigate the impact of increasing, which, theoretically, may reduce the percentage of AGUI records. However, this should be proven experimentally. Moreover, there is a need to find a compromise value for that can reduce the security violation and does not degrade the information usefulness at the same time.

The previously mentioned disruption equations were implements in these two experiments. The equations were detailed in Chapter 4 and formalized as: , where the anonymised blocks are calculated by finding the total summation of the disrupted blocks:. In this chapter, Seer Cancer data was used with some records N= 60,803,185 [139]. The data structure is shown in Table 5.5. Each Q-ID group was anonymised individually, by assigning random values of. Firstly, the three MDSBA services were implemented and experimented regarding the automation of the anonymisation process for different users. Next, the disruption values were calculated for each anonymised G(QID). Each G(QID) was examined with various values of as follows; = {15, 30, 45, 60}. All anonymised records, in both Q-IDs, were examined for AGUI records. Any data record that may re-identify a person with a unique record, and was not anonymised at all, is considered an AGUI record. The AGUI record uniqueness appears across multiple Q-IDs.

AGUI records were detected by transferring the anonymised data to SPSS program [140]. A small SPSS search script was applied to detect the AGUI records, by applying the uniqueness criterion. The criterion is finding any non-anonymised record with no equivalency with any other records. First, user3 account was used, so data was anonymised based on = 60%. The number of AGUI was counted for each value of *k-anonymity*, i.e. for 15, 30, 45, and 60. Figure 5.21 compares between the D level, for both Q-ID groups, and the number of AGUI that appears in all records. The results show a slight increase in D on the larger values of for both Q-ID groups, but however, a dramatic decrease has appeared in AGUI number. This may indicate that AGUI number can be reduced by increasing the value of . This increase of is permitted up to a certain level. Certain level means finding a compromise solution that trade-off between the large number of disruption and the privacy violation. In Figure 5.21, the value of may output less anonymity loss in conjunction with the AGUI reduction impact. These results recall data owners to trade-off between D value and AGUI. This can be investigated by assigning large values of for roles that are permitted to access more than one Q-ID group.

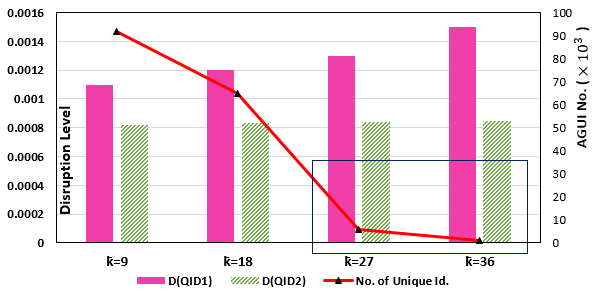


Figure ‎5.21- Tradeoff between D and AGUI No. in []

In the second experiment, the granular access and its impact was tested with the number of obvious guess records. This is essential to identify a proper scale of minimum and maximum values for *k* and values. In this experiment, user1 and user2 were used individually to access each Q-ID group. It is clear that a single group access permission will not face any AGUI problem. However, obvious guess remains an obstacle, even with a single Q-ID group. The anonymised values increase in direct proportion to *k* value, they also, decrease in reverse proportion to obvious guess. In another meaning, the increase value of may decrease AGUI and increase the obvious guess. Thus, one factor increases the D, while another factor decreases it. This is a logical explanation for the slight increase in disruption level, as shown in Figure 5.21.

Figure 5.22 shows that the lower values of have increased the number of obvious guess records. Therefore, choosing an optimal percentage of is essential to compromise a value between low disruption level and low number of obvious guess records. In Figure 5.22, the indicated bars, with a red square, show the optimal range, which is between of 24 – 36. The optimal value in this range can output the lowest values of D and obvious guess. Both experiments show that choosing the proper values of *k* and is the solution AGUI and obvious guess violations.

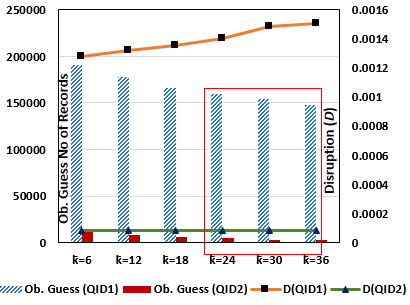


Figure ‎5.22- Single Q-ID group access and obvious guess

## Summary

Big data is prone to multiple user’s access. Hence more fundamental structural shift toward the big data granular access is needed. Few techniques can be implemented to support access granularity. Aggregating Q-ID attributes into separate groups may support the granularity approach. Another technique is the variant numbers of equivalent records. This is implemented by assigning a gradual disparity for *k-anonymity* value. These two techniques are supported in the proposed MDSBA framework. The framework leverages the MapReduce performance and scalability. Its operational steps correspond to MapReduce ecosystem structure. In this chapter experiments, a practical demonstration was conducted to measure the user granular access by applying Hadoop ecosystem tools. The method of granularity is related to *k-anonymity* adjustment, and Q-ID grouping. Hence, users can be assigned to pre-created business roles that are given a maximum percentage of ownership level. We demonstrated permissions assignments in MDSBA. The permissions are managed by creating Q-ID groups, mapping business roles to groups, and delegating roles to organizations. Users are given the access permissions by the delegated organizations.

This chapter demonstrated the three components of MDSBA framework, which are deployed between Federation Service and Service Providers. The three components, of core, initialiser and anonymiser, have shown an automated solution for the granular access control of data anonymisation. The granularity results have appeared on discriminately anonymised data. The contrast between anonymisation levels was calculated by a mathematical equation given as a Disruption D. The experiments showed some security violation on assigning business roles mapped to more than one Q-ID group. The violation appeared across groups as a unique identifier (AGUI). Reducing the impact of this violation requires choosing optimal values of *k* and. It was proven that the small values of *k* may cause a large number of AGUI records, while large value of *k* may cause a high information loss on anonymisation. Hence, we need to find a compromise value of *k* that trade-off between AGUI violation and information loss. Next chapter will focus on finding the best practice on assigning *k* and values for a better security and performance operations in big data

# - Towards Optimal *k-anonymity*

Datasets containing private and sensitive information are useful for data analytics. Data owners cautiously release such sensitive data using privacy-preserving publishing techniques. Personal re-identification possibility is much larger than ever before. Social media has dramatically increased the exposure to privacy violation. One well-known technique of *k-anonymity* proposes a protection approach against privacy exposure. MDSBA adapts *k-anonymity* method by developing a better concept to apply a wider framework that can manage and control the user’s authorization access in a fine-grained approach and a granular concept. This approach may reduce the personal re-identification in a comprehensive framework for big data. However, this may lessen the usefulness of information gained. The value of *k* should be carefully determined, to compromise both security and information gained. Unfortunately, there is no any standard procedure to define the value of *k*. The problem of the optimal *k*-anonymisation is NP-hard [141]. In this chapter, a greedy-based heuristic approach was suggested to provide an optimal value for *k*. The approach evaluates the empirical risk concerning Sensitivity-Based Anonymisation method. The approach is derived from the fine-grained access and business role anonymisation for big data, which forms the framework.

A Quasi-identifier (Q-ID) refers to a subset of attributes that can uniquely identify some tuples in a table. Incautious publication of quasi-identifiers will lead to a privacy leakage. Moreover, choosing a small number of Q-IDs may negatively affect the security, and may leave more chances of re-identifying personal information. Nowadays, re-identifying a person is much easier than ever before. This refers to the internet revolutionary with smart-phones, social media, and services automation. This indicates the risk of the auxiliary information that may lead to identifying a person, and regardless the relationship between the adversary and the victim. This recalls a need for increasing the number of Q-IDs. Any personal information as little as simile should be considered as a Q-ID. The current anonymisation methods assume a limited number of Q-IDs, usually up to seven or nine Q-IDs. MDSBA increases the number Q-IDs and merges them in a small number of attributes, between two to four for each group. Each group is denoted by G(QID), and each G group can be a combination of attributes, as discussed later in this chapter [142].

## Previous Solution to Find the Optimal *k* Value

Eliminating the re-identification through rigorous inference in data is NP-Complete. This is due to functional and multi-valued dependencies. Hence, preventing the privacy violation is eventually impossible [143] [144]. Data is redundant with multi-owner, and the same individual’s private information can be available with multi-owner. Hence, linking released data to other external data may carry hundreds of probable inferences. Moreover, the current social media revolution provides even a higher probability of identifying a person. Knowing few personal information can be facilitated and assisted by searching the individual’s social media public profile on Facebook, Twitter, or Linked-in to identify the age, address, current place, and possibly other personal information. This inference combination increases the privacy violation and derives unlimited possible types of attacks [145]. Data owners may fail to protect all kinds of attacks. Therefore, security advisers need to improve rules and techniques to protect privacy continuously. However, this does not guarantee a complete security protection but may reduce the probability of security threats [146].

The currently proposed *k-anonymity* models do not provide access control frameworks for big data analytics [147]. The core method of any access control is identifying analysers’ needs of data analytics and interests. Researchers implement the de-identification technique that modifies the data such that no combination of quasi-identifiers (Q-ID) smaller than *k* [148]. This technique evolves the organizing of attributes in equivalent groups or domains to gain the *k-anonymity*. However, the key part of the de-identification is assigning proper Quasi-identifiers and the *k* numbers by data owners [149]. Hence, the amount of information gained does not only depend on the method of anonymisation, but also, depends on the chosen number of *k* value, and the way of determining the Q-ID attributes.

There are no any direct instructions to assist business owners in selecting Q-IDs or *k-anonymity* values. Few studies highlight some thoughts about finding the optimal values of *k* and Q-IDs nomination [61]. Contemporary studies have proven that finding optimal *k* value is NP-hard [150]. The hard part of finding *k-anonymity* value is not about identifying any random number, but about finding the optimal known value that is said to be the best. Researchers proposed different techniques to find the best *k* value[62]. Proposed techniques implement either one of two methods; generalising or specializing techniques. Generalization technique suppresses or generalizes all data, computes the cost metrics for finding the best cut, and assigning the optimal *k* value by recursively examining the best specializing level [63]. In the specializing technique, proposed researchers follow the greedy-based heuristic approaches by implementing the entropy equations or by adopting crowdsourcing answers before and after anonymisation [151]. There were few studies suggested techniques for finding the optimal Q-IDs. A distinct Q-ID and a tuple separator are computed to determine the optimal Q-ID and *k* value [137]. However, proposed methods of finding *k* value are expensive solutions, and data owners need to scan datasets numerous times to enable the optimal *k* finding. Also, proposed methods did not study the increasing number of Q-ID attributes on improving the de-identification. A solution for finding the optimal value of *k* in *k-anonymity* method is proposed in this chapter. The solution approach is based on the MDSBA for big data analytics. The framework provides data analytics granularity for multi-domain access.

## MDSBA Grouping and the Gradual Access

This chapter proposes a guidance for data owners on assigning *k-anonymity* parameters. The suggested proposal is related to the role-based anonymisation control framework. The framework provides a fine-grained access control by dividing Q-ID attributes into vertical groups, with two to four attributes for each group. The core method of the granular access is introduced by three approaches; the probability value of Q-ID attributes, the ownership level, and the grouping method of Q-IDs. The probability of Q-ID is calculated by counting the number of unique values in the specified attribute. The ownership level is the key point of distinguishing user’s access levels. The value is an element of *k* in *k-anonymity* method, which determines the number of equivalent records on anonymisation, where 2≤ ≤ *k*.

Related to Definition 3.1 in chapter three, the number of G(QID) groups can be described mathematically, by denoting a number n of Q-IDs, where Q-ID={qid1, qid2, ..qidn} in a table T. The sensitive attributes C are denoted by the number s, C={c1, c2, …,cs}. Each two to four Q-IDs are grouped in a group G. The number of created groups, denoted by ϒ, is related to the total number of Q-IDs (n), and can be presented by Each non-overlapped G(QID) group consists of several Q-IDs and one class, which is usually mapped to one or more business roles R. Let us also denote U as a user, and O as an organization. The role-based anonymisation control is presented as; {G 🡪 R}(many to many relationship), {R 🡪 O}(many to many relationship), and {O 🡪 U}(many to many relationship). As described, users are given permissions to access Q-IDs through their own organizations. Next, a *k* value of *k*- anonymity is assigned to each G group, while is assigned to each organizational R. This means that *k* is a fixed value, and given only once to the G groups regardless user’s privileges, while is a dynamic changing value, and given differently to each role of each organization. Hence, similar role R is given different values for various organizations. To setup the core method of the granular access; data owners create G groups and assign a proper *k* value for each group.

On the other hand, Organizations are delegated some G groups as per business needs. Every organization is given a set of roles with an authorization level of percentage value () for each role. Theis decided based on the service level of agreement between the data owner and the organization, and is given within the range between [0.1 – 1]. The value is calculated as. Organizations are delegated with the requested business roles. It is the organization’s mission to assign its own users to the proper business roles.

It is evident that personal re-identification increases parallel with the number of permitted Q-ID increase. It is believed that any information belonging to individuals may support the personal re-identification. It was proven that the knowledge of additional attributes other than Q-IDs, can also, raise the problem of unique identifiers [152]. Hence, additional attributes are preferable to be accommodated within G(QID) groups. Therefore, MDSBA considers each auxiliary attribute as a potential Q-ID. The proposed technique, of finding *k* value, determines the optimal *k* value used in *k-anonymity* method. In traditional data, finding the optimal *k* value requires many experimental trials before determining the lowest *k* value that leads to the least information loss. In big data, the case is even worse and requires more computation time and costs. In MDSBA, the optimal *k* value is determined with the minimal computation time and costs, as described in section 6.4.

MDSBA method is operated by MapReduce operations. Its approach implements iterated split and filter techniques for data records with mapping, shuffling, and reducing. The first split aggregates data based on sensitive class value C. Each group is anonymised separately. This kind of split supports the parallel operations of MapReduce. The anonymisation then is applied to the lowest probability attribute for each split. For instance, suppose that a G group contains two Q-ID attributes, that is G(QID)={sex, USA\_State}. The probability of sex is P(sex)=0.5, if the values of the attributes are only (male, female), while the P(USA\_State) =1/50=0.02. In this case, the anonymisation is applied by grouping the sex records and anonymising the USA\_State records. The amount of anonymisation applied on sex records is related to the value of , where larger value of promote a higher anonymisation level. Further details about mathematical computations and processes stages was explained in chapter 3 [98].

MDSBA divides the process into four to five stages. The zero stage filters out the *obvious guess* records, as described in section 5.3.5. Stage one filters out the fully equivalent records. All Q-ID attributes must be equivalent so they can be excluded from stage two process. This is stated by [group all(qid)]. The rest of the non-fully equivalent records are further filtered by semi-equivalency in stage two. The semi-equivalency is measured by grouping all Q-ID attributes but the lowest Q-ID probability attribute and stated by [group all(qid)-1]. The lowest Q-ID attribute is anonymised by either taxonomy tree, interval, or suppression. Stage three follows similar steps, by grouping all Q-ID attributes but the lowest Q-ID probability attributes, and stated by [group all(qid)-2]. The least probability attributes are anonymised and merged with the rest of the groups. In the final stage, grouping is only applied to one Q-ID attribute, which is the highest probability one.

## Possible Attacks against MDSBA

Sweeney has addressed few attacks against *k-anonymity* method [16]. The attacks may occur by multiple queries of analytics, so anonymisation is applied on different Q-ID attributes on each anonymisation time. Adversaries may request data several times with multi-queries, so the anonymisation process may apply anonymity on the first Q-ID attribute in the first trial, and on the second Q-ID attribute in the second trial. Hence, linking chance between the two anonymised tables is high.

The previously mentioned attack is not possible in MDSBA. This refers to the consistency of anonymity. In MDSBA, Q-IDs anonymisation refers to the attribute probabilities. The anonymisation always starts with the lowest probability attribute. Hence, amending queries does not switch the anonymisation process to other Q-ID attributes. Also, the class attribute, in MDSBA, must gain the *k-anonymity* equivalency principle. However, other possible attacks against MDSBA can be summarised into two types of attacks; *obvious guess* and Across Groups Unique Identifiers (AGUI). In the *obvious guess*, an adversary may be able to guess the sensitive attribute (class), if the Q-ID attributes are known to the adversary. This violation may appear when a group of equivalent records has one class value. In the previous chapter, it was experimentally proven that increasing the may reduce the obviously guessed records. The *obvious guess* impact may facilitate the security breach and increase the re-identification threat. For instance, if a group of patients has one value of the class ‘Diabetes=positive’ and they share the same race, age, and state, then the intruder can obviously guess the diabetes state of the patient. To avoid this breach, an initial filtration on stage zero is implemented, and before splitting data into class value groups. The *obvious guess* records can be reduced by increasing the value of to a certain extent. However, preventing the appearance of *obvious guess* records is possible by filtering the records in a zero stage, by using UDF program. This resolution was further discussed in section 5.3.7. The second possible attack against MDSBA is caused by splitting Q-ID attributes into small groups. This Across Groups Unique Identifier’s security threat is lower than the uniqueness appearing in each G(QID) group. To launch a successful attack, the attacker needs to know almost everything about the victim’s background. This is believed to be a background knowledge attack. As proven experimentally in the previous chapter, resolving AGUI can be implemented by increasing the value of , so the number of AGUI records decreases. AGUI maximum security threat appears when users are given exactly two G(QID) groups. Giving users more than two groups may reduce the impact of AGUI.

## Finding the optimal *k* value

As discussed previously, finding the optimal *k* value was proven to be NP-Hard. It was also mentioned that small values of *k-anonymity* are inadequate in multi-Q-ID groups. Moreover, MDSBA framework relies on the granular access of users, which requires a gradual increase or decrease in value. A greedy-based heuristic approaches is suggested to find the optimal values of *k*. The obtained *k* value should be as large as possible. We need to consider some factors that may help *k* value estimation. The aim of determining the optimal *k* value is the trade-off between security level and gained information level. We need to be aware of the security risk on assigning a low value of *k*, especially, when datasets contain multiple G(QID) groups. However, larger values of *k* may negatively affect the usefulness of gained information. Therefore, keeping the degradation of data to the minimal level is possible by choosing the optimal value of *k*.

It is obvious that the minimum permitted number of equivalent records is =2. This minimum value achieves the safe threshold value to prevent a unique identification. Based on this fact, the value of *k* can be assigned as per the value. As described before, the minimum permitted value of=0.1. Since = *k*, and the minimum =2, then the value of *k* should be at least *k*=2/0.1=20. Considering the minimum *k*=20, we may increase the value of *k* starting from *k*=20, because dropping *k* below 20 may eliminate the equivalency concept, and ruin the *k-anonymity* principle. In some cases, we may need larger values of *k*, so *k*=20 is the smallest possible value. Datasets with multiple G(QID) groups should be given larger values of *k*. This is referring to the dramatic increase of AGUI values in G(QID) groups. To choose a larger *k* value for each G(QID) group, we first need to ensure that the chosen *k* should not negatively affect the information gained extensively. To do so, let us study the cumulative frequency equation that presents a bench mark or a reference point on measuring the data contrast. The cumulative frequency can be calculated by the disruption equation, as described in the next section.

### Cumulative Frequency

The term frequency in statistics means the number of times a given datum occurs in a dataset. Two types of frequencies are defined in statistics, relative frequency and cumulative relative frequency. The relative frequency measures the fraction of times for a value occurrence. The total percentage of each occurrence must equal to 100%. The cumulative relative frequency represents the accumulation of the previous relative frequencies. The frequency always represents the occurrence of a discrete value or occurrences of the value in an interval of a data. This means all types of frequencies require intervals to find out the frequent values [153].

In datasets, it is possible to calculate the anonymisation loss starting from *k*=20. MDSBA adapts a naïve equation to calculate the anonymisation loss, known as disruption. The disruption equation measures the data disturbance occurred after applying anonymisation. The disruption increases monotonically with the increasing value of *k*. The disruption value was previously described in section 4.5. The equation of calculates one block of anonymised data, where M denotes the number of the records in each block, and N denotes the total number of records for all dataset. The total disruption for all dataset is calculated by the summation of all values of Ɗ[total]. However, this kind of calculation can be a stick in the wheels, since the value of *k* should be calculated before the anonymisation process is conducted, while the disruption equations can only be calculated after the anonymisation completion. To resolve this issue, a rough estimation for the disruption value is calculated before the start of the anonymisation process. The estimation is conducted by finding four checkpoints of *k*.

To assign the checkpoints; data owners need to define the *k* value range first. This depends on the security level of the internal organizational policy. Data owners may choose a range between [20 – 80], or even higher. Data owners, then choose the four checkpoints, which are {}. The symbol of represents the first value of the *k* range, which is =20. The second symbol of represents the last value of the *k* range, which can be =80 or even higher. The initial suggested *k* value is =20. The data owner needs to investigate the new value =30. If the CF of the new value is far away from the reference line, then *k* will remain 20. To save the calculation time for disruption values, it is better to count the number of non-equivalent records when *k*=20, and then when *k*=30. This rough calculation accelerates the process of finding the optimal *k* value.

Practically, finding the number of non-equivalent records are straight forward steps in Pig Latin script. A fast computation script aggregates all equivalent records,

*SG= GROUP data by (QID1, QID2, QID3)*

and then filters the grouped records

*frequency= FILTER SG by k < 20*

The number of non-equivalent records for each value of *k* is accumulative. Hence, the cumulative frequency (CF) can be calculated based on the number of non-equivalent records. To state the CF mathematically, let us denote the disruption values for {} by { respectively. The values of represent the lowest and highest disruption for minimum and maximum *k*, while the values represent the current and new inspected disruption for the current and new inspected *k*. Eventually, the number of inspected values of *k* are denoted by *I*. The cumulative frequency is measured by disruption values as:

(1)

The inspected is approved if the 0.9 ≤ CF ≤ 1.1. A range of 0.2 is allowed for data dispersion in comparison to the linear increase of data disruption. This equation is formulated to reduce the information loss that occur because of anonymising data with larger values of *k*. To demonstrate Equation 1, let us study the Seer data records, which concludes three G(QID) groups with classes in bold, as shown in Table 6.1.

A data owner has decided an interval of *k* between [20 -80], so the seven inspected values of *k* are {20, 30... 80}. It is apparent that *k* value increases by 10’s units to avoid the overhead computation cost. The initial value of *k* is (20), while the inspected value for the three G(QID) groups is *k*=30. The lowest disruption value for G(QID)1 is, the disruption is calculated based on the lowest probability attribute, which is (Age=0.012). Recalling Equation 1, the 0.93. Since the CF ≥ 0.9, then the new value of *k*=30 is accepted. Next, we need to inspect *k*=40, so 0.54, which is < 0.9. Therefore, the new *k* value is rejected, and the value of *k* remains as *k*=30. Figure 6.1 illustrates the CF of the G(QID)1, and the diversion when *k*=40 from the reference point. The reference point is found by calculating the linear disruption increase based on the minimum and maximum D values only.

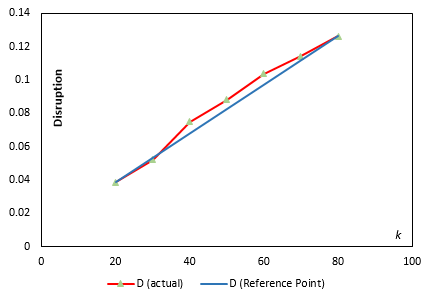


Figure ‎6.1- Cumulative Frequency for Seer G(QID)1 disruption

Table ‎6.1- Seer dataset with three G(QID) groups

|  |  |  |
| --- | --- | --- |
| G(QID)1 | G(QID)2 | G(QID)3 |
| Age | Year | CS Tumor |
| Sex | Race | Radiation |
| County | Grade | Marital Status |
| State (class) | Diagnosis (class) | Survival months (class) |

### Linear Regression

Regression Analysis is another statistical modeling that is related, in somehow, to the frequency distribution. This type of modeling helps understanding the relationship between an independent variable with a regular increase, and dependent variable changes. The regression analysis is being used for prediction and machine learning [154]. In MDSBA, linear regression analysis may give a general model for the anonymisation loss. In anonymisation concept, the regression analysis can be applied to measure the level of anonymisation loss. It is known that increasing the *k* value will decrease the number of fully-equivalent records. This, in turn, will increase the anonymisation loss, and as a result, will reduce the gained information. In another word, the anonymisation loss increases parallel with the *k* value increase. The *k* value can be considered as an independent value, because it increases regularly without depending on any other variables. The anonymisation loss depends on the *k* value, which means it is a dependent variable that changes increasingly parallel with the *k* value increase [155].

The previous cumulative frequency equation can be used in archived data to determine the optimal *k* value. The same equation can also be used in live data, if accuracy was ignored. Live data increases with the time, and the disruption values keep changing accordingly. The disruption value decreases proportionally with the increase number of data records. The previous CF equation is just a naïve equation derived from the linear regression. Finding the first and last disruption values to draw the regression line is an easy task but inaccurate. More accurately method is drawing a regression line and finding out the largest diverted values. The regression line is found by the equation, where, and. Replacing the anonymisation parameters imposes that the disruption line *d* replaces, and *k* replaces. The general regression equation is given by:

(2)

The value of the slope is formulated as:

(3)

(4)

Equations 2, 3 and 4 are used to calculate the regression line for the disruption values. This equation can be used in both archived and live data, to determine the largest disruption values in a specific range of *k* values for a certain size of the dataset. However, this equation is intensely needed in live data, since the regression line keeps changing parallel with the data growth. More data records impose a disruption decline with a minor disruption disparity between the various *k* values. For instance, if the disruption value for *k*=30 is *D*=0.001, then the disruption value for *k*=40 may be D=0.012. This small disparity value between two values *k* may lead to a high error rate, if the previous CF equation was used. Hence, using a more accurate equation is strongly recommended to determine the optimal value of *k*.

In the previous example, of applying CF on Seer data has resulted of *k*=30. The diagram of Figure 6.1 has shown a quite large disruption when *k*=40, hence, the successful value was *k*=30. If the regression line was replaced by CF, then Equations 2 and 3 calculate the *d* value as shown in Table 6.2. The values of D represent the actual disruption values, which are found by grouping and investigating the number of anonymised records. The regression lines of d and D values are plotted in Figure 6.2.

Table ‎6.2- Seer data / G(QID)1 results after line regression calculation

|  |  |  |  |
| --- | --- | --- | --- |
| ***k*** | ***d (reg. line)*** | ***D*** | ***D-d*** |
| 20 | 0.040879 | 0.03865 | -0.00223 |
| 30 | 0.055709 | 0.051998 | -0.00371 |
| **40** | **0.070539** | **0.074965** | **0.004426** |
| 50 | 0.085369 | 0.088056 | 0.002687 |
| 60 | 0.100199 | 0.103597 | 0.003398 |
| 70 | 0.115029 | 0.114283 | -0.00075 |
| 80 | 0.129859 | 0.126033 | -0.00383 |

As noticed in Table 6.2, similar results to CF are found when *k*=40. Table 6.2 shows the subtraction result of (*D-d*). The table shows the largest contrast value between the regression line and the actual line, when *k*=40. It is essential to remember that if the *D* values are dropped below the regression line are accepted, as a reason of their low disruption. Table 6.2 shows some negative values when *k*={20,30,70,80}. The largest disruption value must be positive. The largest value prevents *k* from gaining a higher value. In the previous example, the largest disruption value has occurred when *k*=40. The initially chosen value was *k*=20, then moved up to *k*=30. The *k* value progression was inhibited when *k*=40. Table 6.2 shows similar results when *k*=40.

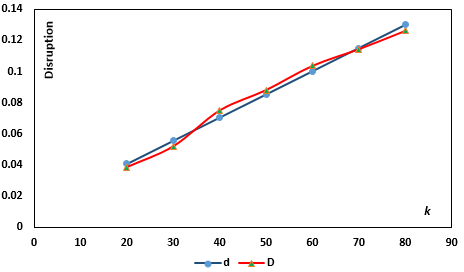


Figure ‎6.2- Regression calculation for Seer G(QID)1 disruption

#### Three security levels

In linear regression, some disruption values are below the regression line, while others are above the regression line, as shown in Figure 6.2. This is apparent because the regression line presents the average middle line between various disruption values. Suppose that three disruption values fall above the regression line. The axiomatic question whether we consider the highest unique value as an inhibitor, or any one of the three highest values can be considered as inhibitors. For instance, if we study the disruption values for G(QID)2, as presented in Table 6.3. We find that the highest three disruption values appear when *k*=20, 30, and 60. The first two values have close similar values. In such cases, it is better to leave this for the data owner to determine the number of inhibitors. More inhibitors will hinder the *k* value progression, which means gaining smaller values of *k*, and as a result, less security applied.

Table ‎6.3- Seer data / G(QID)2 results after line regression calculation

|  |  |  |  |
| --- | --- | --- | --- |
| **Seer G(QID)2** | | | |
| ***k*** | **d** | **D** | **D-d** |
| 20 | 0.02468 | 0.027145 | 0.002465 |
| 30 | 0.03674 | 0.039215 | 0.002475 |
| 40 | 0.0488 | 0.048745 | -5.5E-05 |
| 50 | 0.06086 | 0.058712 | -0.00215 |
| 60 | 0.07292 | 0.076587 | 0.003667 |
| 70 | 0.08498 | 0.082143 | -0.00284 |
| 80 | 0.09704 | 0.098473 | 0.001433 |

High security imposes a higher value of *k*, so choosing one inhibitor can be used when data owners apply a high-security level on a specified dataset. Generally, we can decide three different levels of security {high, medium, and low}. Data owners may decide to go with any of the three security levels. In addition, other security options, such as the *k* value interval may support the security levels, as mentioned earlier. Briefly, we may summarize the security definition applied to any datasets by creating Table 6.4.

Table ‎6.4- Security levels options setup by data owners to decide the optimal k value

|  |  |  |
| --- | --- | --- |
| **Security-Level** | **Number of inhibitors** | ***k* value interval** |
| High | 1 | Example [20 – 200] |
| Medium | 2 | Example [20 – 150] |
| Low | 3 | Example [20 – 80] |

In Seer / G(QID)2 example, if data owner chose the security level=high, then the value of *k* for G(QID)2 is (***k* = 50**). This is because the highest distribution value was found on *k*=60. For the high-security level, we use only one inhibitor, which is *k*=60. Therefore, *k* should stop on the value before the inhibitor. If the security level=medium, then the value of *k* is (***k*=20**) because *k*=30 is the second inhibitor. Also, similar *k* value is gained for the medium security level.

## Finding the Optimal *k* percentage

It is important to remember that *k* value is given for each G(QID), and it is not related to user’s access and privileges. Regardless user’s access, *k* value is assigned to G groups, while is given as per user’s access level and privileges. For this reason, we need to identify a reasonable approach to assign the values for each permitted G(QID) group. The given value of can be determined based on the service level of agreement with organizations. However, a minimum value of should be determined to avoid security violation. Thewas clearly explained in the previous chapter/ section 5.2.2. The *k̄* is calculated as *k̄* =. Apparently, more permitted G(QID) groups manifest a higher personal violation probability. We need to investigate if this is correct, by experimenting various dataset groups. Experiments were conducted to measure the percentage of Across Groups Unique Identifiers (AGUI) appearance. The aim is to identify the factors that increase the AGUI records. This is essential to identify the best approaches for assigning the values. Two datasets of Adult and Seer were anonymised. Each dataset concludes three G(QID) groups as shown in Tables 6.1 and 6.5. The three G(QID) groups are mapped to two business roles (HR Manager, and Oncologist). In each dataset, the HR Manager is permitted to access G(QID)1 and G(QID)2, while the Oncologist is permitted to access G(QID)2, and G(QID)3, as shown in Figure 6.3. In the first experiment, an anonymisation was applied on G(QID)1, and G(QID)2, which are mapped to (HR Manager). Every anonymised G(QID) was measured by disruption for different values of

Table ‎6.5- Adult dataset with three G(QID) groups

|  |  |  |
| --- | --- | --- |
| G(QID)1 | G(QID)2 | G(QID)3 |
| Age | Relation | Gain |
| Job | Race | Loss |
| Marital | Sex | Hrs-per-wk |
| Edu (class) | Work-class (class) | Salary (class) |

In the second experiment, an anonymisation was applied on G(QID)2, and G(QID)3, and measured by disruption equation for differentvalues. In the third experiment, a user was given both roles; HR Manager, and Oncologist. The number of AGUI records are counted after completing the anonymisation. The anonymisation was applied with several values for both roles. The results are shown in Figures 6.4 and 6.5 for Seer and Adult datasets sequentially. The figures indicate that increasing the number of G(QID) groups will increase the disruption, which contributes to reducing the number of AGUI records. G(QID) groups, in both datasets, were given different values of *k*, range between [20 – 50]. The experiments show two factors may support the reduction of AGUI; the increased value of, and the increased number of G(QID) groups. However, the number of permitted G(QID) groups is related to the assigned roles, and we have no control over it. Thus, we can control the minimum value of . Users with one G(QID) permission group can be given any preferred value with no restrictions. Moreover, the highest AGUI appearance occurs when the number of G(QID) groups=2. With the continuous G(QID) group’s increase, the number of AGUI keeps declining. This approach leads us to increase theto one, when the number of groups=2. Figures 6.4 and 6.5 show that AGUI number is reasonable when), and G(QID) groups are two.

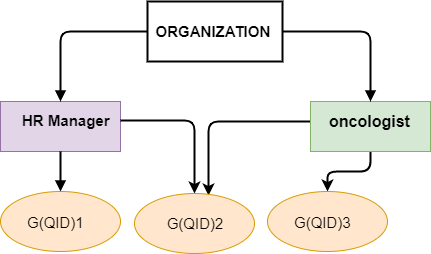


Figure ‎6.3- Groups mapping to business roles

The experiments showed that should be assigned based on the number of G(QID) groups. Referring to Figures 6.3 and 6.4, if an organization has been granted only Oncologist role with two G(QID) groups, then is fair enough to reduce number of AGUI to the minimal. If another organization was given both roles with three G(QID) groups, then is adequate to reduce the number of AGUI. The value of can be given to any role that is mapped to more than two G(QID) groups. However, the number of G(QID) groups is one factor, denoted by F (QID) that affects the estimation. Business owners may determine their own factors to conclude a comprehensive equation that estimates. In this thesis, only two factors are defined; the number of G(QID) factor, denoted by *f(QID)*, and the organizational trust level *f(T).* The *f(QID)* was explained earlier, as it is assigned by data owner. The *f(QID)* is given the value 1, when the number of G(QID)=2, and given the value less than 1, when the number of G(QID) > 2. The factor *f(T)* is given a value within the range between [0.1 -1], where 0.1 is the most trusted organization for the data owner, while the least trusted organization is given the value of 1. The final is found by calculating the average value of both factors. The final equation is. However, data owners may decide their own equations based on their own security policy.

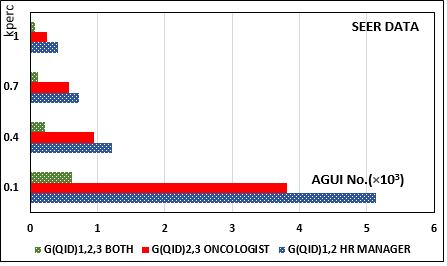


Figure ‎6.4- AGUI Num. for both roles in seer data

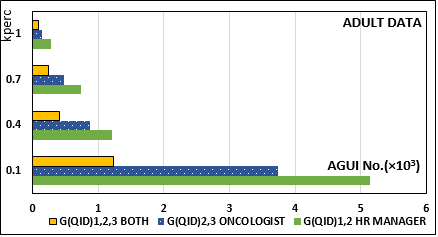


Figure ‎6.5- AGUI Num. for both roles in adult data

In the previous example, if an organization has requested two business roles of Oncologist and HR Manager, and *f(T)*=0.2, then *f(QID)* can be given a value of 0.8, i.e. *f(QID)*=0.8. The final average is given by. In another example, if the same organization has requested only Oncologist role, then *f(QID)* can be given a value 1. This is because the two G(QID) groups manifest the highest AGUI value, hence,. Suppose that G(QID)1 is given *k*1=30, and G(QID)2 is given *k*2=40. The is calculated for the Oncologist by 1=300.6 = 18, and 2=400.6 = 24. If a user has requested an access with the Oncologist role, then a pair of values will be produced to authorize the user.

## Dynamic G(QID) groups

In some datasets, there are many attributes available in data schemas, while user’s inquiries and interests vary. In MDSBA, it is possible to control the amount of data that is permitted for organizational access. Permitted data may consist of a small or large number of attributes. The G(QID) groups are pre-determined by data owners before mapping them to business roles. Roles later are delegated to organizations. Organizations need to look at each business role and its related mapped G(QID)’s, so they can decide the best-fit roles for their business nature. One of the obstacles that organizations may face, is the rigid G(QID) groups, where organizations really worry about attributes rather than groups. If, for instance, an organization requested only one attribute from a G(QID)1, and another attribute from G(QID)2. In this case, we may need to define new G(QID) groups for each organization, which is impractical. For this reason, dynamic G(QID) groups are preferable solutions, which suit different flavours of organizational queries.

To explain this approach, let us consider a set of attributes Vi for a dataset S= {V1, V2, V3, V4, V5, V6, V7}, let G(QID) groups are determined by data owner as the following G(QID)={G1(V1, V2, V3, V4 ), G2(V5, V6, V7)}, and the attached business roles to the G(QID) groups are R={R1(G1), R2(G2)}. Suppose that an organization has chosen to analyse the following attributes Ś= {V3, V4, V5, V6}. It is possible to enforce the organization to follow the G(QID) groups, but it will be more reliable if we have pre-defined G(QID) groups for such cases. The convenient solution can implemented by creating different patterns. Each pattern concatenates different attributes and assign them to separate business roles. In the previous example, we may define a second pattern for G(QID) group= {G3(V3, V4, V5, V6 )}, and we may map the second pattern to R={ R3(G3)}. However, the third role (R3) should be exclusive to the second pattern, so each created pattern should have its own roles. Data owners may create two or three patterns for each dataset, and organizations may trade the best suit attributes. Next, the software application should be able to find the closest match pattern to the user’s choice.

One of the real examples that can be implemented by dynamic patterns, is a census and survey data. Census data consists of hundreds of attributes, and creating multi-pattern is the desired option for analysers. One of the census examples is available in USA Ipums; the site provides many samples of data collected by survey or census [156]. Some datasets were downloaded from Ipums website, with the attributes shown in Table 6.6. The table shows chosen data after creating G(QID) groups and classes. The pattern divides attributes into five G(QID) groups, with one highlighted class for each group.

Table ‎6.6-Pattern 1 of census data presented by G(QID) groups

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PATTERN 1** | | | | |
| **G(QID)1** | **G(QID)2** | **G(QID)3** | **G(QID)4** | **G(QID)5** |
| REGION | FRIDGE | SCHOOL | NO\_CHILD | **EMPSTAT** |
| COUNTY | PHONE | HIGRADE | **RACESING** | LABFORCE |
| CITY | FUELHEAT | EDUC | BIRTHPLACE | LOOKING |
| **HOMELAND** | **VEHICLES** | GRADE\_ATT | YR\_IMMIG | WORKEDYR |
|  |  | **DEGFIELD** | SPEAKING |  |

The above attributes, also can be re-grouped in different patterns, and this depends on the user’s queries and demands. Table 6.7 shows another suggested pattern that data owners may follow.

Table ‎6.7- Pattern 2 of census data

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PATTERN 2** | | | | |
| **G(QID)6** | **G(QID)7** | **G(QID)8** | **G(QID)9** | **ATTRIBUTES** |
| N\_CHILD | HOMELAND | SCHOOL | REGION | FRIDGE |
| EMPSTAT | BIRTHPLACE | HIGRADE | COUNTY | PHONE |
| **RACESING** | YR\_IMMIG | EDUC | CITY | FUELHEAT |
|  | **SPEAKING** | GRADE\_ATT | LABFORCE | VEHICLES |
|  |  | **DEGFIELD** | **LOOKING** |  |

It is necessary to establish separate business roles for each pattern. This is essential to avoid *k* value conflict between G(QID) groups. Organizations may choose the required attributes, and an automated algorithm can choose the best fit pattern. For instance, if an organization decided to analyse the following attributes (REGION, COUNTY, LABFORCE, LOOKING, EMPSTAT). The automated decision can determine the best pattern by creating a comparison matrix. The matrix lists all available patterns, and calculates the availability of each attribute within the G(QID) groups. Table 6.8 matrix determined that pattern 2 is closer to the user’s query.

Table ‎6.8- Matrix for choosing the best pattern

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | PATTERN 1 | Availability Perc. | PATTERN 2 | Availability Perc. |
| **REGION** | G(QID)1 | 0.5 | G(QID)9 | 0.8 |
| **COUNTY** | G(QID)1 | 0.5 | G(QID)9 | 0.8 |
| **LABFORCE** | G(QID)5 | 0.75 | G(QID)9 | 0.8 |
| **LOOKING** | G(QID)5 | 0.75 | G(QID)9 | 0.8 |
| **EMPSTAT** | G(QID)5 | 0.75 | G(QID)6 | 0.33 |
|  | TOTAL | 3.25 | TOTAL | 3.53 |

The matrix calculates the availability percentage of each attribute and chooses the highest total pattern. The availability percentage is computed by dividing the number of attributes appearances in each G(QID) group over the total number of attributes for the specified G(QID) group. For example, the availability percentage for (REGION and COUNTY), in the first pattern, is the result of two attributes appeared in G(QID)1 over four attributes, which is the total number of G(QID)1 group attributes, so 2/4= 0.5. Similarly, G(QID)9 is calculated by counting the number of appeared attributes (REGION, COUNTY, LABFORCE, LOOKING), while the total number of G(QID)9 attributes are five, and the division result is 4/5= 0.8.

Practically, the matrix can be stored in database and presented as a part of entity relationship diagram, as shown in Figure 6.6. This relationship diagram is part of the dataset created for the core service as described in section 5.2.3.

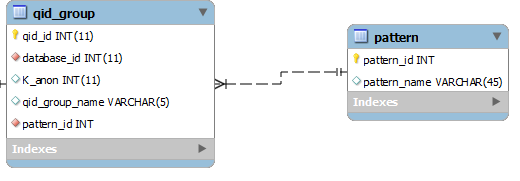


Figure ‎6.6- Part of ERD showing the relationship between patterns and G groups

MDSBA core service contains a program that can pick up the most relevant pattern. Table 6.8 matrix is used to calculate the highest pattern score. The matrix can be presented in an algorithmic structure by reading each pattern from the database, reading all related Q-ID groups, and reading all attributes attached to each Q-ID group. Each G(QID) group is stored in a three-dimensional array, which stores the attributes and pattern number. The array presents: the pattern number is denoted by *i*, the G(QID) number is denoted by *j,* and the attribute name is denoted by *k.*  For the three values, there is a need for three loops program, to read each array object for calculation. For instance, in the previous example of the census, two patterns were created by the data owner. Each pattern consists of a set of G(QID) groups. An iterated algorithm counts the number of the available G(QID) for census dataset. In the previous example, the number of the available G groups is 9, so 9 arrays are stored in a three-dimensional. The array presents three values of; G(QID) number, pattern number, and Q-ID description as:

*array\_G\_QID [i][j][k]={(1,1, REGION), (1,1, COUNTY), (1,1, CITY), (1,1,HOMELAND) ,….,(2,9,REGION), (2,9,COUNTY), (2,9,CITY), (2,9, LABFORCE), (2,9,LOOKING)}.*

Next, the chosen attributes, by users, are stored in a separate array. The array is iterated and compared with the three-dimensional array. With each comparison, mathematical calculations are conducted based on the number of objects match between both arrays. A complete algorithm is shown in Figure 6.7.

|  |
| --- |
| **Algorithm for finding the best fit pattern** |
| *Input: user chooses the needed attributes*  //the chosen attributes by users  // counts the number of patterns for database\_id =1111  //Loop for each pattern  //read all Q-ID IDs from the specified pattern  ”  //Loop for each G(QID)  ”  // Create three-dimensional array for all patterns  (*k*))  /\* The array is created and now it is ready for matching\*/  //the first loop for the patterns available  //The second loop for all groups in each pattern  //Number of Q-IDs in each G(QID)  //Loop for user’s attributes  //If statement to match attributes  //Number of equal objects between G(QID) and user’s attributes  //Calculate the probability between equal objects and the actual Q-IDs |

Figure ‎6.7-Algorithm for determining the best patterns

## Summary

A major issue with *k-anonymity* is known related to finding *k* value. Finding the optimal *k* value was proven to be NP-hard. The previously proposed methods of finding *k* value heuristically are computationally expensive. Data owners need to experiment with various values of *k* to determine the best *k* value that concludes the highest information gain. These methods are even harder in big data anonymisation. MDSBA provides a greedy-based heuristic approach to find the optimal values of *k*. The approach suggests an initial *k* value with a possibility to increase the value gradually based on the cumulative frequency (CF) of the data. CF equation is practically cost-effectively computation method, and can provide an approximate *k* value before commencing the anonymisation process. Moreover, Linear Regression is another computation method to find the value of *k* accurately. CF is a special case of Linear Regression, where Linear Regression concludes a reference line to calculate the optimal *k* value with a reasonable computation time.

The second part of the chapter is determining the access granularity, by choosing the best ownership level. This is implemented by dividing Q-ID attributes into small groups, and mapping the groups to business roles. Organizations, then, are delegated to the required roles with an authorization level of for each role. Theis determined by different factors. The experiments unveil two factors; trust percentage between data owners and organizations, and number of permitted G(QID) groups. The experiments recommended a value of 70% for, if the number of G(QID) groups are larger than two, and a value of 100% if the number of G(QID) group is two. The proposed method of finding helps reducing Across Groups Unique Identifiers (AGUI), while the proposed approach of finding *k* values helps reducing *obvious guess* risk.

The last part of this chapter is creating various G(QID) patterns by shuffling the Q-ID attributes as per organization’s desires. When the number of Q-IDs is quite large, then customers may choose different Q-ID attributes. Since access permissions are assigned on the G(QID) level, then users are permitted to access some unneeded Q-ID attributes. This randomness may create false access privileges, by allowing users to access unneeded attributes. To avoid such a security breach, several random G(QID) are created. The pattern is chosen automatically based on the chosen attributes by organisations. To do so, a matrix is created to calculate the best value for the number of appearances. The pattern is chosen as per the highest score, which is concluded by the most number of appearances.

# - Comparison between MapReduce and Spark

One of the biggest concerns of big data and analytics is privacy. It is believed that the forthcoming frameworks and theories will establish several solutions for the privacy protection. One of the known solutions is the *k-anonymity* that was introduced for traditional data. Recently, two major frameworks leveraged big data processing and applications; these are MapReduce and Spark. Spark data processing has been attracting more attention due to its crucial impacts on a wide range of big data applications. One of the predominant big data applications is data analytics and anonymisation. The previous chapters discussed a complete framework for data anonymisation. The anonymisation framework was merely implemented for MapReduce processing framework. This chapter will set-up a comparison between MapReduce and Spark in processing performance of data anonymisation. Spark is a fast processing framework that was implemented in several applications such as; SQL, multimedia, and data stream. The main focus of this chapter is the SQL Spark, which is adequate for big data anonymisation. Since Spark operates in-memory platform, we need to observe its limitations, speed, and fault tolerance on data size increase. Spark introduces an abstraction called resilient distributed datasets, which reads and serializes a collection of objects partitioned across a set of machines. Developers claim that Spark can outperform MapReduce by 10 times in iterative machine learning jobs. The experiments in this chapter compare between MapReduce and Spark. The overall results show a better performance for Spark’s processing time. However, in some limited cases, we prefer to implement the old MapReduce framework, when the cluster resources are limited and the network is non-congested [157].

## Analytics in Big Data

Big data evolution has formed new software tools and techniques to accelerate the processing speed, and increase the scalability. The new tools targeted many big data applications such as data analytics. The analytics has manifested some security concerns, as a reason for big data publicity prominence. In general, big data is beneficial when it is public, this means many organizations from different fields need to access this data for multiple purposes [92]. They all analyse, mine, and output statistical results. However, exposing any private data to public view carries a high-security risk. Personal re-identification is the main focus of researchers since decades. In data analytics, adversaries can easily re-identify and violate some private information. The information may contain sensitive information about patients, bank agents, census, or any other private information [128].

Anonymisation methods, based on *k-anonymity*, have been widely employed to prevent data re-identification [13]. Anonymisation methods fall into two broad categories. The first category constitutes of techniques that generalize data from the bottom of the taxonomy tree towards its top and are referred to as the Bottom-Up Generalization (BUG) [127]. The second one is based on walking through the taxonomy tree from the top towards the bottom, known as the Top-Down Specialization (TDS) [23]. TDS and BUG methods were mainly developed for traditional data. Therefore, researchers upgraded the old methods to suit the new operations of big data. The operations should consider the parallel and distributed processing steps. Various methods of anonymisation were specifically designed for parallel distributed processing. However, most resolutions fall short of a proper parallelization capability. The reason for this is further explained in section 7.6. Some of these recently developed anonymisation methods are; Parallel BUG [24], Hybrid BUG and TDS [25], and Two-Phase TDS [26].

The previously mentioned methods focused on developing algorithms to find the best cut of the taxonomy tree, the optimal values of *k*, and/or the best option of anonymisation technique either by TDS or by BUG. Implementing these algorithms may require high computation costs of continuous iterations with conditional statements, which means multiple times of heavy scan for the whole data records. However, these algorithms have ignored two main facts about big data processing; firstly, the key success factors of parallel processing is a proper parallelization algorithm [94]. This can be achieved by reducing the iteration to the minimal possible level. This is essential to avoid multiple scans for large data records. The major concern is not only the time consumption, but the unexpected failure that arbitrarily occurs during big data processing. Secondly, the contemporary methods have ignored the changes that occur in data growth. With the increased number of data records, data gain more similarity in attribute’s values. This is apparent in our life’s activities. For instance, if we sit in a data hall with 100 people, and the probability of finding a person’s age=33 is 10%, then the probability of finding the same age may go up to 20%, If the data hall contains 1000 people. This is because people’s age range is between [0 - 100], so more people will definitely increase the value equivalency. This concept was discussed in section 3.1.1.

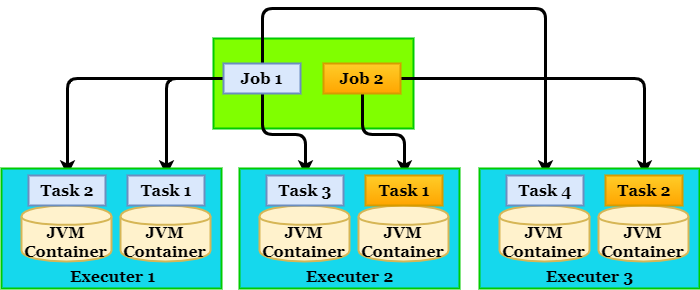
The previously mentioned facts are actually essential to understanding big data nature and specifications. Applying heavy computation to a certain group of data records to find out the best anonymisation cut, or even to decide which attribute that we need to anonymise, is inadequate. In big data, applying such accurate techniques may not affect the final results of statistical output. We even may ignore the small statistical value of small decimal results. The statistical results follow the principle of estimation prospect, which gives data miners a flexibility of approximating and rounding some numbers [28]. Therefore, pre-calculating the *k* value, and pre-determining the attributes needed to be anonymised is an advantage. Generally, this non-accuracy will not dramatically affect the data analytics results.

MDSBA is an anonymisation method that uses BUG in *k-anonymity* that can address the previous two facts about big data. The main aim of this method is to improve the anonymisation performance and to increase the usefulness of anonymised data. MDSBA is not only an anonymisation algorithm or technique, but it provides a fine-grained access control for multi-level of user’s permissions.

In this chapter, experiments are conducted by using two different big data analytics tools, MapReduce and Spark. Anonymisation operations will apply *k-anonymity* in MapReduce framework and compare it with Spark. Spark is an in-memory cluster computing framework for processing and analyzing large amounts of data. It exploits the increased size of hardware resources in CPU and RAM. Nowadays, Spark is the most popular processing framework for big data, by providing cost-effective and high scalable processes. MapReduce and Spark are both popular open source cluster computing frameworks. These frameworks are used in big data for large-scale data analytics, by applying parallel distributed processing tasks. Both frameworks provide programming API to users on managing major components of mapping, shuffling, execution, and caching [158].

## Spark Structure

Spark operations are different from the traditional MapReduce. Spark architecture is implemented to increase process performance. For this reason, multiple jobs can run in parallel by implementing; applications, executers, and active drivers. The traditional MapReduce splits each job into many tasks, and each task is processed by a single process within each container, so the process terminates when the task is completed. In Spark, each node contains one or more executers, and each executor operates in one container. The executor is a JVM base, as shown in Figure 7.1. The node may have many containers, which depends on the node capacity. Each container comprises one executor process that can run multiple tasks, and it remains for the life-time of the Spark life. This structure accelerates the initiation of the process and the tasks. Also, Spark consists of a process, known as active driver. This driver is used to manage the job flow and schedule tasks, and it is located on the master node. It interactively communicate with the executor of each node. If Spark was deployed on the top of YARN, then Spark driver can run over the cluster [159].



‎7.1- Spark structure and jobs distribution

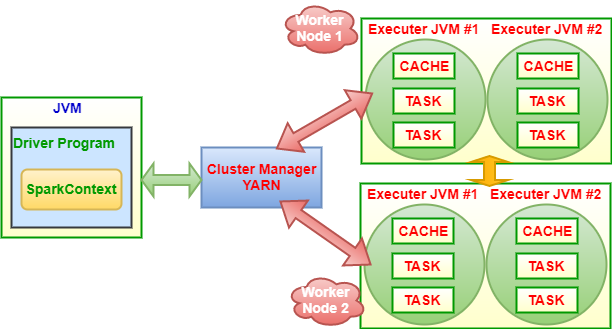
Spark is shipped with distribution processes that are able to mimic the functions of YARN. Users may install Spark in three different modes; stand-alone, cluster mode, and client mode. In stand-alone mode, the built-in resource manager is able to manage the cluster nodes, and without a need for YARN or Mesos. Both cluster and client modes are deployed over YARN. However, each mode was designed for different tasks. Users may run cluster mode for production jobs, while client mode is used for user interactive and debugging jobs. The main difference between both modes is the location of Spark driver [157].

One of Spark negatives is the programming difficulties that programmers may face. Spark operates in RAM, and programming with large data may derive Spark running out of heap memory. This is because of the unnecessary RDD data collection caused by the programmer algorithm. Programmers should have previous knowledge about Spark core structure and jobs like partitions, nodes, serialization, JVM, executors, memory and disk, shuffles, compressed files, and columnar formats (parquet). Therefore, they may need to try various algorithms to deduce the most efficient one. This frustrated and time consuming code may cause bugs within the program execution as soon as the data exceeds the maximum limit of resources. Usually, cached data that do not fit in memory are either spilled to disk or recomputed on the fly when needed, as determined by the RDD's storage level. However, this does not prevent data growth bugs and over-flow [160].

Anonymity in data analytics is an example of complex analytics, were anonymisation operations scan the data records many times during the filtration, aggregation and masking operations. The anonymisation processes latency is considerably high, therefore, batching tools are more efficient to deal with the large data size, and long latency. The big data tools were developed to accommodate both of data batches and streams. The first generations of MapReduce frameworks, such as hadoop, were unable to process the data stream. The next generation were developed based on Lambda architecture, which is designed to handle both batch and stream processing methods. The framework structure attempts to trade-off between latency, throughput and fault-tolerance. Most of the, recently developed, real-time frameworks follow similar structure of storing temporary data frames and tables in the temporary random memory, so most of the operations are completed without performing input/output operations thereby decreasing latency [128].

Spark consists of masters and workers. The master uses the cluster manager to acquire the worker's resources. The cluster manager can be Mesos, YARN, or Spark standalone. Since Hadoop v2, YARN was developed with a wide range of flexibility, so it is able to collect information about the cluster resources and pass it to Spark, which in turn sends the command back to the cluster manager to create a certain number of JVMs or executers on each node [160]. Users create RDDs by applying operations called transformations and actions. The actions triggers the lazy transformations. The transformations such as; map, filter, and groupBy cannot operate without proper actions such as; collect, count, and saveAsTextFile. Spark exposes RDDs through a functional programming API in Scala, Java, Python, and R, where users can simply pass local functions to run on the cluster. Spark may perform 2.5, 5, and 5 times faster than MapReduce [161].

In Spark two types of JVM are created, one is created on the master for Spark driver to contain the context operations, and the other type is created for each worker. Each worker may contain more than one JVM, and each JVM is known by a core. The worker executer may contain one or more cores. The number of cores on each worker can be determined automatically by the system, or statically configured by the user. Each Spark core is created by initiating a JVM, which reserves a specified size of memory space. The memory is reserved for; heap memory, cache, and JVM processes. Each core runs multi-task at the same time. The complete Spark structure is illustrated in Figure 7.2.



‎7.2- Spark structure in master and workers

Spark operates as JVMs in master and workers. The main connector between Spark master and workers is the cluster manager, such as YARN. On submitting a job by users, Spark in master, known as Driver Program, acquires resources from the cluster manager. The cluster manager collects information from workers, such as disk space, memory size, and processors. The driver Program splits the job into multiple tasks as per available resources. The small tasks are scheduled and executed accordingly. The execution is scheduled and divided into stages of DAG. The tasks are scheduled on the executors using a low-level scheduler provided by a cluster manager. The executors run the tasks submitted by Spark in parallel. To empower the fault tolerance structure, each core operates in an independent JVM to isolate the misbehaving JVM from the others. This structure is essential to reduce the dead nodes possibilities. However, this structure may become a cumbersome in some cases, where interaction between two processed data in two separate JVM’s is not possible without storing each executer data on the disk. This behavior degrades Spark efficiency and speed [122].

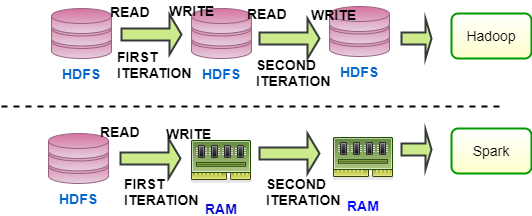
After processing each task, the results are either returned to the driver program JVM or proceeded to the shuffling. Every core can run multi-task concurrently. All Spark cores reserve similar partitions in the memory, in order to copy data blocks for execution. The size of partitions depends on the memory size. The number of executed tasks depends on the number of partitions created in each core. By default, Spark creates one task per data partition, therefore, programmers have options to decide the number of partitions on each task within the core. For a better parallelization, the number of partitions should consider many factors, such as the number of executers on each worker, the executer memory size, and the HDFS data block. Each partition reads one or more of data blocks for processing, so if HDFS data block is 128MB, while the number of partitions in the whole cluster is 32, then a data size of 8 GB will be split into 64 tasks. That is 32 partitions can load 32 blocks with 32 tasks, and each block is 128MB. Therefore, the data size of (8192 MB / 128 = 64). Moreover, more partitions can help the parallel processing of data. However, more partitions requires larger memory size [162].

## MapReduce and Spark

Both frameworks are very similar in some core features. Spark runs on Hadoop, Mesos, or standalone, hence it is not possible to categorize Spark as a non-MapReduce framework. MapReduce core structure is YARN and Hadoop Distributed File System (HDFS). These two Hadoop native processes are intensively used in Spark. They provide reliability, performance, and scalability for Spark. It is worth mentioning that the distinguished difference between MapReduce and Spark is the processing methods. MapReduce wastes a considerable time on input/output transmission between memory and disks. The inefficiency of read/write from the disk and the high latency in each operation are the major inhibitors in MapReduce. On the contrast, Spark operations are executed over the built-in memory, and without a need for read/write on disks [157].

Before applying Spark on MDSBA, some concerns need to be verified about the efficiency of Spark in anonymising data. Anonymisation requires a large buffer size to accommodate the massive size of data flow. The comparison between MapReduce and Spark, tends to evaluate MDSBA approach’s performance in each framework. Also, it is essential to correlate between MDSBA architecture and Spark operation. This correlation measures the capability of MDSBA to operate on Spark framework. Moreover, all previous chapters on MDSBA were conducted in MapReduce environment, and within Hadoop ecosystems. Other frameworks have not been tested so far. There is a considerable number of cluster computation engines available in the market during the last decade. This includes; Hadoop, Spark, Flink, Storm, Heron, and Samza. Storm is a real-time processing engine that is able to stream SQL records from Hadoop Distributed File System (HDFS), from Kafka, or from MongoDB, or Redis [33]. We chose Spark over Storm for the following reasons: Storm is a popular framework worldwide, and it is implemented in enterprise company’s networks like Yahoo, Twitter, Flipboard, and yelp. It is written in Clojure Language, the Lisp-like functional language. Moreover, the structural design of Storm was built for the data stream, which means continuous infinitive processes to analyze data. Non-streamed data, or batch data, cannot be processed by infinitive processes. Anonymisation methods are only applied to batch data. This is essential for grouping, counting the number of equivalent records and masking the non-equivalent. These operations cannot induce adequate anonymisation results in the data stream. Hence, Storm is not preferable in *k-anonymity* methods [161].

Pig is an old Hadoop ecosystem that operates in MapReduce structure. Spark is a new framework that gained popularity for its speed and performance. Spark is much faster than Hadoop in performing operations. The speed is related to the in-memory processes. On the other hand, Hadoop inputs/outputs from/to the disk on starting or completing each map/reduce process. Hadoop reads data from the disk, executes in two stages of Map and Reduce, and returns output back to the disk. This data flow of in/out creates a high input/output latency, and keeps the scheduler busy with creating more tasks. This may make Spark faster than MapReduce, which may reach up to a hundred times faster. Speed is an important issue in big data for its large size. This new framework opens the door for various new applications that was not possible to develop with the old frameworks.



‎7.3- Comparison between Hadoop and Spark in dealing with memory and disks

Figure 7.3 illustrates a comparison between Hadoop and Spark. Spark’s in-memory cluster computing capabilities are high, which boost performance, even with the large data magnitude. The time difference between reading data from disk or from memory is significant. Larger data size shows higher latency than smaller data size when reading from disks. Also, Spark implements caching technique to store data in memory to minimize the disk I/O. Figure 7.3 clearly describes the iterated stages in Hadoop operations. That is keeping two stages mapping or reading from the disk, and reducing or writing to the disk. A considerable time is consumed between memory/disk input/output. This two-stage principle is not available in Spark, instead, it maps blocks from disks, processes them, keeps them in-memory for another staging process. Hence, several stages may save the disk I/O.

The second reason that makes Spark faster is the advanced job execution engine. Both Spark and MapReduce convert a job into a DAG of stages. The graph theory is an old theory that represents any graph with a collection of vertices connected by edges [163]. Graph theory was developed and implemented in many fields such as; computer science, and medical science. In Hadoop and Spark, graph theory is being used for scheduling tasks, with a similar technique to DAG. Each job is presented by a direct graph, which is a set of vertices connected by directed edges. Each vertex in the graph represents one processor, and each edge represents a communication link. The difference between Spark DAG and Hadoop DAG is the communication link. In Hadoop, each processor reads all incoming messages from the in-edge (disk), performs some computation, and writes messages to the out-edge (disk). The in-edge and out-edge are presented by disk I/O, which cause major delays. Hence, two stages can be processed at a time. In other words, there is no way you can start at some vertex in a DAG and follow a sequence of directed edges to get back to the same vertex [164]. Hadoop creates a DAG with only two pre-defined stages, map and reduce. Developers are forced to process their commands within one of these two stages. Based on this core structure, complex jobs need to be split into two or more jobs to fulfill the two-stage processes. In contrast, Spark can accomplish the job with multiple stages, and without splitting the single job into many sub jobs. This structure optimizes job scheduling and processing. For instance, if the processing algorithm contains, read from storage, filter, and group. Spark can accomplish this algorithm in one job with three stages of the map, shuffle, and reduce. In Hadoop, this will be divided into two jobs of (map, shuffle), and (map, reduce) for each job. Data will be read from the disk, filtered, and stored back to the disk with the first job. Again, data will be read from the disk, grouped and stored back to the disk.

However, some obstacles may degrade the efficiency of Spark. As mentioned earlier, if data size does not fit the total memory size of the cluster, then Spark uses disks to spill data. This action causes a higher delay on disk I/O, which is a similar delay reason in Hadoop framework. Therefore, Spark cluster always needs a large memory size. The average amount of processed data should be considered on building Spark cluster. This is because Spark has the capability of caching RDD in memory. Developers benefit from this feature to avoid disk I/O. However, caching features requires enough quite large memory size. For instance, if Spark cluster is expected to analyze data size of 3 TB, then Spark uses up to 60% of the total configured executor memory to cache RDDs. If data analysers decided to cache half of the 3 TB in-memory at once, then there should be enough memory space to accommodate this large size of data. In this scenario, a Spark cluster with 20 workers and 128 GB memory for each worker is enough to cache 1.5 TB. A rough calculation can be. As noticed, Spark cluster infrastructure should consist of powerful servers and fast connectors between workers and storage units. Therefore, virtual machines connected to storage nodes by iSCSi or Fibre Channels are not the proper choice for building Spark cluster. Spark requires physical or non-virtual servers. Also, network connections between servers and storages should be through direct access connection (DAC), such as SAS or SATA.

Spark allows applications to cache data in memory for processing. This enables applications to minimize disk I/O. A MapReduce-based data processing pipeline may consist of a sequence of jobs, where each job reads data from disks, processes it, and writes the results back to disks. Since Spark allows caching of data in memory, the same application implemented with Spark reads data from disks only once. Once data is cached in memory, each subsequent operation can be performed directly on the cached data. Thus, Spark enables an application to minimize I/O latency, which can be a significant contributor to overall job execution time.

The (**persist**) or cache commands in Spark can be done automatically or by developers. Spark persists some intermediate data in some shuffle operations without users calling persist. This is done to avoid re-computing the entire input if a node fails during the shuffle. However, it is recommended for developers to call persist on the resulting RDD or during the map phase [165]. A common misconception is that Spark cannot be used if input data does not fit in memory. This is not true because Spark can process terabytes of data on a cluster that may have only 100 GB total cluster memory. However, this is not recommended for data anonymisation by SQL Spark. The small memory size compared to the data size may degrade Spark operation’s performance. It is up to the developer to decide what data should be cached and at what point in a data processing pipeline. In fact, if a data processing application makes only a single pass over data, then there is no need for caching data at all. Developers may cache data, when data is read several times [128].

Data lifecycle is rotated between the network, disk, and memory. Data format and structure differ between disk storage, memory, and network transmission. The process of converting data to a storable format, in disks, is known by serialization, while releasing data in reverse structure, in memory, is known by de-serialization [166]. Data can be serialized using readable or binary format. The readable format may include CSV, XML, JSON, while the binary format may include multimedia and database. In Big data, implementing text file for storing and retrieving data in a text format is inefficient in terms of either storage space or parse time. The binary formats are more compact and can be parsed much quicker than text formats. However, in this paper we have limited resources in our conducted experiments, therefore, the dataset sizes we use are quite small, so there is no major impact regarding data type.

## Data Streaming versus Batch

Data should be stored in repository storage units. Data analytics operations fetch data from the repository and process them. Stored data may represent a production data or archived, where the production data is a live data with a real-time or close to real-time update. Archived data is an old data that is no longer actively used. From the analytics perspective, archived and backup data are similar. The difference between backup and archived data can be controversial, since the technical definition for each data type is slightly different. The archiving is applied to the original data, while the backup is a copy of the data. However, the aim is identifying the analytics operations on real-time and non-real time data. Regardless the originality of data, real-time data needs to be analysed differently [167].

Anonymity in data analytics is an example of complex analytics, where anonymisation operations scan the data records many times during the; filtration, aggregation and masking operations. The anonymisation latency is considerably high, therefore, data batching tools are more efficient to deal with the large data size, and long latency. Various big data tools were developed to accommodate both batching and streaming data. MapReduce framework, Hadoop, is unable to process the data streaming. The next generation was developed based on Lambda architecture, which is designed to handle both batching and streaming data. Lambda structure is based on a trade-off between latency, throughput, and fault-tolerance. The recently developed frameworks follow a similar structure of storing data temporarily in the memory, to reduce the disk input/output latency [168].

## Implementing MDSBA in Spark

Spark has many advantages over Hadoop ecosystems. It mitigates latencies, and increases the performance. For instance, Pig divides jobs into small tasks, and for each task, Pig reads data from HDFS, and returns data back to HDFS once the process is completed. This in/out consumes a considerable time, and unlike Spark, which implements Resilient Distributed Dataset (RDD). RDD is the main distinguishing feature of Spark. RDD divides jobs into many DAG stages, and for each stage, Spark reprocesses RDD in memory without referring back to the disk. Spark may perform many times faster than MapReduce.

In this chapter, Pig and Spark performance are evaluated in MDSBA. To conduct the detailed evaluation, Spark’s transformation and functions, used in anonymisation, should be properly defined. In this chapter, all Spark scripts are coded by Scala program. Some functions and transformations operate faster than the others. For example, inner **join** transformation commands may require a high computational cost. MDSBA was proposed for big data processing frameworks. Its core concept is applying optimized anonymisation procedures and algorithms by splitting data into small tasks, so they can be parallelized among the cluster nodes.

In big data processing, the reduce phase is expensive since it involves data partitioning, data serialization and deserialization, data compression, and disk I/O. These operations require data transfer over the network to aggregate data over multiple nodes. Leveraging any application’s behaviour should consider the size of data transferring between nodes during the reduce phase [162]. In data anonymisation, reduce phase is presented by SQL grouping commands, which causes a high shuffling processes. In order to reduce the *groupBy* impact in the anonymisation application, a filtration command is initiated to split data logically as per nominal values. This type of split reduces the performance degradation caused by shuffling process. This reduces the amount of shuffling among cluster nodes during the reduce stage.

The previous anonymisation techniques, such as Bottom-Up Generalization (BUG) and Top-Down Specialization (TDS), were supposedly to work efficiently in the parallel distributed processing. Technically, anonymising data with these algorithms may negatively create data overflow without considering the cluster resources and capabilities. For instance, Figure 7.4 illustrates a comparison between TDS and MDSBA algorithms. In TDS algorithm, the data flow may negatively affect parallelization. Firstly, grouping all records without filtering data are inefficient. This was experimentally approved in section 7.6. Secondly, the rest of operations should be implemented with the help of UDF. Spark and Pig are unable to run such intensive computations and conditional iterations. Eventually, the anonymisation algorithm cannot complete the operations successfully without having more advanced methods. Scripting languages such as; Pig and Scala are not able to complete such intensive operations. Therefore, UDF) is embedded in the script’s codes to execute the complicated operations. However, the operations of UDF are executed in a black box, and not related to Spark or Hadoop control and management. This is also true in Pig operations. UDF uses the resource of locally installed JVM, and does not process data within RDD. Therefore, implementing Spark with such an algorithm is inefficient.

In MDSBA, UDF was embedded in Spark and Pig scripts, with the minimal size of data processing. As explained in the next section, the anonymisation is applied on fewer attributes at a time. This technique controls and minimizes the size of data flowing to the UDF. The UDF is executed in a local JVM beyond the source manager’s control. It is a memory pool located outside Spark JVMs. Therefore, there is a need to reduce the amount of data flowing outside Spark JVMs [160]. Moreover, the anonymised Q-ID in TDS is volatile. This means that the specialization is applied to different Q-ID attributes in each group of records. The chosen attribute, for specialization, is the one with the highest score value [23]. Calculating the highest attribute score for each group is an expensive computation process. On the contrast, in MDSBA the anonymised Q-ID is pre-determined based on the Q-ID probability. This saves a considerable amount of computation time. This solution may not provide the optimal anonymisation for the pre-chosen Q-ID, by resulting in a higher percentage of anonymisation loss. However, we may sacrifice some information gain to the benefit of data performance.

**Group** all records

Generalize all records

**UDF**

**UDF**

**Group** the highest QID probability

Anonymise the lowest QID prob.

**Group** the highest QID probability

**UDF**

YES

STOP

Anonymise the lowest QID prob.

If non-equivalent

**Filter** as per class value

If more cut is available

If non-equivalent

**Group** all records

NO

YES

Specialize the highest QID score by choosing the best cut

MDSBA

TDS

YES

‎7.4-Comparison between the anonymisation algorithms of MDSBA and TDS

Figure 7.4 compares between TDS and MDSBA in anonymising data. The TDS algorithm imposes various iteration on calculating the best cut and scores. This type of iteration is inefficient in big data, and even worse in using UDF, where the program executes the UDF code outside Spark framework. Furthermore, the UDF program needs to iterate large size of arrays. The UDF executes almost all anonymisation process. Thus, there is no real benefit from the parallel distributed system. In contrast, MDSBA implements UDF with a limited data size flow. As shown in the figure, using UDF was reduced to the minimal operations. UDF reads only few data attributes to apply some masking operations. He data size flowing to the UDF is relatively small.

### User Defined Function in MDSBA

MDSBA implements User Defined Functions (UDF) in different locations. This is essential for two main purposes; anonymising and ungrouping. In anonymising, three masking types of interval, taxonomy tree, and suppression are implemented. Figure 7.5 shows the algorithm for anonymising any numerical group. In Scala, the group of objects can be a list or a sequence. In the shown algorithm in Figure 7.5, one attribute of a numerical type was used for anonymisation. Minimizing the amount of data on accessing the UDF program is essential to reduce the processing cost and to avoid data overflow as described before.

It is hard to predict the failure of non-Spark 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 re-configured and enlarged if needed. If Spark worker memory is large enough to fit the data size, then the external JVM that handles the UDF may be able to handle up to 25% of data size located in Spark. 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 anonymise data with the minimal number of iterations.

Section 3.2.4.4 in chapter 3 described the interval masking algorithm in details. The algorithm principle is simple, which converts a numerical list of values into set of intervals. Each intervals consist of a minimum and a maximum value. In MDSA, the minimum values are found by rounding the minimum number in the list down to the nearest 5. The round down is different from the ordinary rounding. As shown in Table 7.1, the values of the example are rounded down to the nearest 5.

Table ‎7.1- Round down to the nearest 5 example

|  |  |  |
| --- | --- | --- |
| **Number** | **Round down to the nearest 5** | **Math. calculation** |
| 2 | 0 | 2 – 2 mod 5 |
| 6 | 5 | 6 – 6 mod 5 |
| 12 | 10 | 12 – 12 mod 5 |
| 14 | 10 | 14 – 14 mod 5 |

The round down is calculated by using MOD 5. The round value is only used as an initial value. In the interval, the minimum value is calculated by round down, while the maximum value is calculated by adding the interval range to the minimum value. For instance, if the interval range is 10, then the values of (4, 7) can be presented by a range of [0 – 10[. The initial value of 4 is 4-4 mod 4=0. The maximum value is 10 + 0 = 10. However, the value 10 is not included in the interval, instead it is included in the next interval of [10 – 20[. The interval range is derived from the sensitivity level ψ. If ψ=0.5, then interval range = 1 / 0.2 = 5. The interval range must be a multiple of 5, so if the number in the interval range was found to be smaller than 5, then it is rounded up to 5, and if the number was found to be larger than 5, then it is rounded up to multiples of 5. This concept is described in Table 7.2. The example shows a list of ψ values and the interval range calculation.

Table ‎7.2- Interval Range Rounding down to the nearest 5 example

|  |  |  |
| --- | --- | --- |
| sensitivity level ψ | Interval Range | Interval Range Rounding |
| 0.4 | 1 / 0.4 = 2.5 | 2.5 +(5 – (2.5 mod 5))= 5 |
| 0.015 | 1 / 0.015 = 66.6 | 66.6 + (5 – (66.6 mod 5)) = 70 |
| 0.045 | 1 / 0.045 = 22.2 | 22.2 + (5 – (22.2 mod 5))= 25 |

Table 7.2 illustrates an example for calculating the interval range round. The equation used in calculation can be mathematically presented as:

(7.1)

It is possible to round the both of minimum and maximum value of the interval, while keeping the interval range. In this case, the minimum value is rounded down, and the maximum value is rounded up. However, rounding up the interval range or the maximum value conclude similar results.

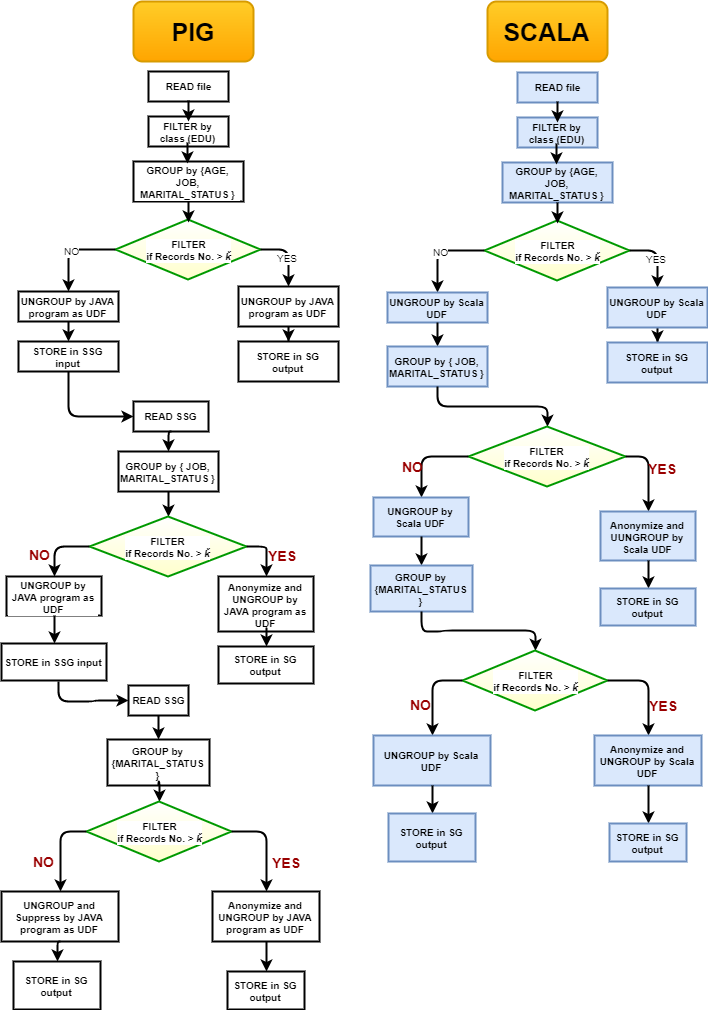
In a similar algorithm, we can mask data with taxonomy trees as explained in section 3.2.4.3. The aim of this algorithm is reducing the size of data flowing to the UDF program. This is implemented by masking fewer attributes at a time, then attaching the rest of the data tuples, to the anonymised attributes. To find out the estimated size of data flow to the UDF, We may roughly estimate the size of ten data records by 1KB, while the size of one attribute of the ten records may not exceed the 20 bytes. This shows that the actual data size flowing to the UDF file may not exceed one-third of the total data size. For this reason, data processing in UDF is not expectedly expensive.

### Differences between Pig and Spark Algorithms

Figure 7.5 shows two algorithms implemented by Pig and Spark. Both are quite similar with minor differences in the data flow between memory and disk. Spark does not need a disk read/write operations. In both programs, the fully equivalent records (SG) are stored in disks as output files. The major difference between both programs is the semi-equivalent records (SSG) and their management. In Pig, SSG records are stored temporarily in disks. Moreover, Pig is not provided with a built-in UDF capability, therefore, an external program, such as Java, should handle the UDF operations. Unlike Spark, which provides a UDF built-in capability by defining a Scala function.

Pig is a script used for MapReduce framework, which executes any task in two stages only, map and reduce. Hence, every action requires a temporary storage in disks. In contrast Spark does not require such limited stages. It uses memory for a temporary cache for all stages. Spark can complete one task and move to the next task by creating a series of map and reduce. Each map and reduce process is known as a stage. This is very efficient technique, but requires enough memory to cache each stage in memory. If the memory does not fit the output size of data, then the memory spill the rest of data to the disk.

In Spark algorithm, it is difficult to concatenate the anonymised data with the rest of the attributes in the table. For instance, if a table T contains the following attributes of T={A,B,C,D}. The anonymisation process has finished masking the attribute A, and transformed it to a, that is (A 🡪 a). Hence, the anonymised attribute {a} needs to be concatenated with the rest of the attributes {B,C,D}, so the new table TA will be TA={a,B,C,D}. In Scala's implementation of DataFrames, there is no direct way to zip two DataFrames into one. We can simply work around this limitation by adding indices to each row of the data frames. Then, we inner join DataFrame by these indices. Appendix 3 presents a full Scala program including UDF.



‎7.5- Anonymisation algorithms in Pig and Spark

However, all join operations are known as Cartesian join, which require high number of shuffling between nodes. Therefore, join operation is very expensive, and it is not recommended by Spark developers. The right solution is implementing another way of concatenating without creating an independent DataFrame. To understand the concept of this UDF script, let us give the following example, four Q-ID attributes and one non-Q-ID attribute, Teacher, are presented in Table 7.3. On grouping the records by three attributes (CLASS, SCHOOL, and LEVEL). The rest of the attributes, TEACHER and MARK collect multiple values in arrays. The script calls the anonymisation UDF to anonymise the marks, and concatenates the rest of the attributes in one table. This can be implemented through (withColumn) command. The command syntax is;

*val anonymise \_MARK =Grouped\_QID.withColumn("MARK", AnonUDF($"MARK")).select("TEACHER","CLASS","SCHOOL","LEVEL",”MARK”).*

Table ‎7.3- Grouped and anonymised table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Q-ID Group | | | |
| **TEACHER** | **CLASS** | **SCHOOL** | **LEVEL** | **MARK** |
| Jones, Khan, Steve | A | School 1 | 1 | [70-80[, [70-80[, [70-80[ |
| Mark. Jane | B | School 2 | 2 | [90-95[, [90-95[ |

Table 7.3 should be ungrouped for storing or possible further processing. This format of grouped records cannot be easily managed for computation or statistical operations. Ungrouping the grouped data can be accomplished in various ways. The best-found way was creating another UDF that is able to map every sequence to a wrapped array, and rotate the direction of the wrapped array. The aim is converting Table 7.3 to Table 7.4 format. It is clear that the three Q-ID attributes are repeated according to the number of grouped objects in MARK and TEACHER.

Table ‎7.4- Illustrates the new state after ungrouping records

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Q-ID Group | | | |
| **TEACHER** | **CLASS** | **SCHOOL** | **LEVEL** | **MARK** |
| Jones | A | School 1 | 1 | [70-80[ |
| Khan | A | School 1 | 1 | [70-80[ |
| Steve | A | School 1 | 1 | [70-80[ |
| Mark | B | School 2 | 2 | [90-95[ |
| Jane | B | School 2 | 2 | [90-95[ |

The Ungrouping algorithm reads each wrapped array, counts the number of objects, and maps each array with indices. Each wrapped array has a various number of objects, therefore we need to define a function that can update the array size on each wrapped array. Scala defines functions by using (val) or (def) command. In our case, we implement (def), so the command can update the number of array objects, with the following syntax;

**def assertSameSize(arrs:Seq[\_]\*) = {assert(arrs.map(\_.size).distinct.size==1,"sizes differ")}**.

This definition is called by the UDF that is able to rotate the direction of the arrays with the following command:

*val ungroup = udf((xa:Seq[String],xb:Seq[String],xc:Seq[String],xd:Seq[String],xe:Seq[Integer]) => {*

*assertSameSize(xa,xb,xc,xd,xe)*

*xa.indices.map(i=> (xa(i),xb(i),xc(i),xd(i),xe(i)))*

*})*

The above UDF ungroups Table 7.3, and expands the wrapped array to the format of Table 7.4. In this example, the anonymisation UDF outputs the range of marks in one string of values MARK={[70-80[, [70-80[, [70-80[}. This string should be converted to a wrapped array before ungrouping it. Converting a string to an array, in Scale, is implemented by the command split (col(MARK)). As noticed, implementing the fastest algorithm relies on several trials of execution before choosing the best method. In general, programming in big data should be carefully considered. This is quite similar to programming multi-task programs on a single computer with multiple processors. The program may not gain any advantage of the multi-core processor without a proper algorithm. For instance, an operation of total = a + b + c + d will run in one core only, while the rest of the cores are in ideal states. The same operation can be completed faster, if the algorithm was amended by tot1= a + b, tot2= c + d, and **total**= tot1 + tot2. Splitting the single operation to three operations of tot1, tot2, and total, enhances the performance and leverages the parallel processing on multiple cores. However, the operation of the **total** will be completed on one core only. The final result of the **total** will wait for the parallel operations of tot1 and tot2 to be completed. This operation is known as a sequence operation, which causes the major delay in algorithms [94]. Applying similar concept to parallel distributed operation mimics the mapping and reducing operations. Reduce is a sequence operation that waits for the mapping completion, and before the shuffling operation start-up. More shuffling leads to a higher operating cost.

The operation of the grouping process is implemented by the built-in transformation command “groupBy”. Alternatively, Scala permits the SQL embedded commands, so the grouping can be implemented in either way of groupBy or Select query. However, groupBy was found to be more efficient in performance wise. Each data tuple or record contains many attributes. As described before, MDSBA creates small Q-ID groups, which includes two to four attributes only. Also, each data record may consist of several G(QID), classes, and non-Quasi attributes. For this reason, we need to groupBy each G(QID) independently, while the rest of the attributes must be aggregated and concatenated with the anonymised G(QID).

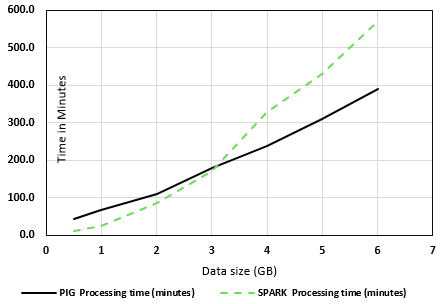
## Comparison between Hadoop ecosystems and Spark

Experiments were conducted to compare between Hadoop ecosystem and Spark. The experiments aimed to measure the performance of the old MapReduce framework and the new Spark framework. The performance includes the computation cost, and scalability of each. Since Spark is a distributed in-memory platform, we need to observe Spark’s behavior on data growth. Eventually, Spark was developed for the new powerful servers that are provided with a large size of memory, and a considerable number of CPU cores. Spark is a memory consumer and CPU intensive operator, therefore, each worker should contain a reasonable size of memory and cores. Usually, memory size in each worker starts from 16 GB, with two quad-core processors. However, the required size of memory and processor in each worker of the cluster depends on three main factors, these are; the data size, the time required to complete the job, and the number of workers and masters within the cluster.

A lab was setup our in Western Sydney University, which includes five virtual machines, with one master and four workers. Each node’s CPU is a single core Intel(R) Xeon(R) CPU @ 2.40GHz, with a physical memory of 8 GB, and the operating system is CentOS 7. Both Spark and Hadoop were setup in the same cluster. Spark 2.1 was setup on Apache Hadoop 2.7. Also, Pig was setup in the NameNode to run the Pig Latin script. We created two different scripts programmed in Pig Latin and Scala. Both scripts must output similar results. To save resources, we executed Pig script first, then Scala script after the completion of Pig script. Adult data was used for the experiments. Data were randomly enlarged up to seven different sizes, and these are 500 MB, 1 GB, 2 GB, 3 GB, 4 GB, 5 GB, and 6 GB. The size of data sets was chosen related to the limited available resources in our cluster.

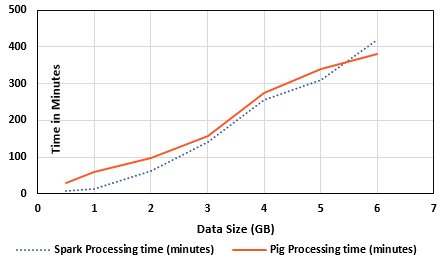
The first experiment aimed to compare the processing time between Spark and Pig. Both scripts are designed to read the data files, filter data by class value, group, anonymise and ungroup. The program concept is similar to the algorithm described in Figure 7.5. The adult data attributes include {AGE, JOB, MARITAL\_STATUS, EDU, SOCIAL, RACE, SEX, POSITION, COUNTY, COUNTRY, SALARY}. Attributes are divided into three groups of Q-IDs. The first Q-ID group is G(QID)1={AGE, JOB, MARITAL\_STATUS ,EDU}, where EDU is the class attribute. The second group is G(QID)2= {SOCIAL, RACE, SEX, POSITION}, where POSITION is the class attribute. And finally, the third group is G(QID)3= {COUNTY, COUNTRY, SALARY}, where SALARY is the class. For this experiment, group G(QID)1 is anonymised, by grouping Q-IDs of {AGE, JOB, MARITAL\_STATUS} in the first stage, then {JOB, MARITAL\_STATUS}, in the second stage, and finally { MARITAL\_STATUS} only in the third stage. Spark was setup with a total of three workers with 6 cores. Each worker node contains two cores, and each core contains 2 GB of memory. The three nodes have 12 GB of memory, with a total of 6 executers or cores.

The results with 3 workers show a large contrast in processing time between both scripts of Scala and Pig. Figure 7.6 shows the processing time for various data sizes. Spark is much faster than Pig in the relatively smaller data size. The relativity is described by comparing the data size with the memory size. In the proper memory size, Spark may reach up to 8x faster than Pig. On data size growth, Spark performance degrades dramatically, so the processing time becomes similar to Pig on data size= 3 GB. More data growth shows better performance for Pig speed in comparison to Spark. Actually, Pig does not speed up processes on data growth, but its processing time grows up steadily. In contrast, the discrete line in Figure 7.6 shows an exponential growth for the Spark processing time. SQL Spark consumes more processing time, when the memory is not large enough in comparison with the data size. For instance, when data size = 6 GB, the process time was around 570 minutes, which is much larger than Pig process time, which was around 390 minutes.



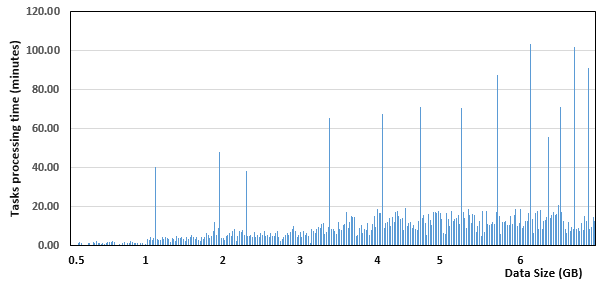
‎7.6- Comparison in process-time between Pig and Scala scripts with 3 workers

In the second experiment, one more worker was added to observe Spark’s behaviour with the larger hardware capacity. The worker node consists of 8 GB memory and two core processors. The previous experiment was conducted by 3 workers and one master, while this experiment was conducted by 4 workers instead. The fourth worker was added to both Pig and Spark clusters. It showed a dramatic increase in Spark performance, which was expected. Figure 7.7 illustrates the same datasets with the extra worker added to Spark domain and Pig domain. Spark’s processing time showed a slight degradation, when data size exceeded 5 GB. However, the overall performance was better than using three workers only.



‎7.7- Comparison in process-time between Pig and Scala scripts with 4 workers

MapReduce is prone to network congestion during map or shuffle phases. Spark reduces this negative impact by reducing the data transmission between disks and memory. This feature is essential in the transmission of the network. Having many trials of experiments with Pig; several tasks took much longer time before throwing errors and terminating the tasks. The engineering structure of MapReduce and Spark is similar when handling the slow tasks. They both implement a speculative execution, which tags any task that takes longer on average than the other tasks from the same job. It clones this slow task and runs it on another node. It will not stop the slow task, but rather run another copy in parallel. This is beneficial in large clusters, whereas small clusters may lose their available resources. However, the university network is not dedicated to MapReduce structure, and suffers from a high network congestion most of the day. In both cases of enabling or disabling Pig’s speculative execution, almost similar delay has appeared in some tasks. Therefore, each experiment was repeated several times, mainly, in running Pig script. The failure jobs where repeated and excluded from the comparison time. The failure percentage of processing tasks in Pig script was around 4%, which increases with the data growth. Many factors may cause this congestion, such as university virtual environment and connection types between storages and virtual machines. Big data clusters rely on direct access connection (DAC) between nodes and storages. Also, the virtual environment is not recommended for big data structure. Figure 7.8 shows 319 tasks that were executed by Pig script for various data size. The tasks belong to more than 15 jobs with an average processing time between 10 seconds to 18 minutes for the successful tasks. The shown pulses represent the failure tasks after long processing time. The failure tasks increase in parallel with the data size increase.



‎7.8- Failed tasks in Pig script are shown in a high processing time

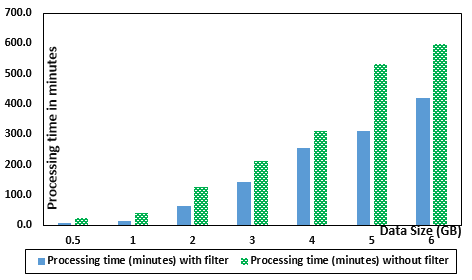
### Spark Tuning in MDSBA

Tuning Spark is one of the hardest tasks on managing Spark cluster. This requires knowledge, experience, and several experiments. There are no clear instructions on tuning Spark to gain the best performance. The focus here is on anonymisation operations only. Different tasks and applications may require different steps of tuning. In MDSBA, the main operations conclude grouping, ungrouping, and masking data. These three main operations should be organized properly to give the best possible performance. Both ungrouping and masking require UDF programs. In masking, the algorithms should consider the least number of iteration and the smallest size of data. As explained earlier regarding the importance of reducing the data flowing to the UDF. Several tuning techniques were experimented and configured. Two main tuning concepts are found; these are: filter/group, and cache data.

In SQL grouping, experimentally it was found that filtering data, and then grouping it, may reduce the grouping time and enhance the performance. Hence, we need to filter and then group data, rather than jumping over to the GROUP command first. It is apparent that grouping data records can replace the command FILTER, so technically, we can group any records without the need for filtration first. However, to reduce the number of shuffling times and leverage the parallelization, it is better to filter records first. For a better understanding, let us consider a set of passenger’s records as shown in Table 7.5. If the grouping command included the attributes of; Depart, Arrive, and Flight, then the grouping results can be {ATL, DXB, (Adult, Child), K380}. We also gain similar results if we use filter command first. For instance, if we filter the records with the class value =K380, then data size will be reduced to two records instead of three. Using filter/group commands concludes the same results as using group command alone. However, in performance-wise they are not similar.

Table ‎7.5- Passenger’s records

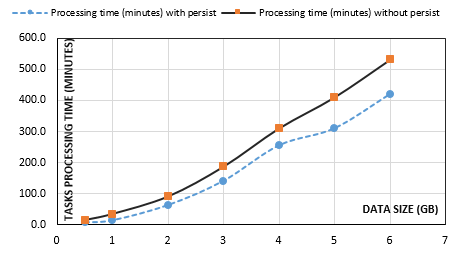
|  |  |  |  |
| --- | --- | --- | --- |
| **Depart** | **Arrive** | **Passenger Type** | **Flight** |
| ATL | DXB | Adult | K380 |
| HND | SYD | Adult | D120 |
| ATL | DXB | Child | K380 |



‎7.9- Process time between filter/group and group only

Figure 7.9 shows the processing time difference between grouping data after filtration, and grouping data without filtration. It is clear that filtering data first increases the program performance. Filtering data first is beneficial, when the attribute values are known, so they can be added by programmers. In MDSBA, the attribute values are pre-added automatically in early preparation stages. MDSBA framework consists of three main services; core, initialiser, and anonymiser. The Scala script is generated in the initialiser stage by reading dataset parameters and user‘s access parameters from two different XML files. User’s XML file is received from the federation service, while dataset XML file is previously uploaded by data owners to the service provider’s servers, parallel with the dataset. MDSBA framework is further explained here [169].

The second tuning in Spark is data caching. The percentage of the failure tasks in Spark script was much lower than in Pig script. The percentage was even lower, when using persist command on reading data from disks. As mentioned before, the (**persist)** command caches the data in memory. Developers may assign the (**persist)**, if data will be read many times for multiple tasks. The comparison between (**persist)** and non-persist commands showed a tangible difference in performance between them. Persist command was implemented after reading a dataset from the disk. Figure 7.10 shows a large difference between both cases, where using (**persist)** reduces the processing time of tasks. The (**persist)** command is part of Spark tuning to increase the overall performance efficacy. However, the command is not recommended when the data size is larger than the available memory. If the data size is larger than the memory, then the overloaded memory will be spilled to the disk. Figure 7.10 illustrates the performance comparison between (**persist)** and non-persist. The time processing contrast increases with the increase of the data size.



‎7.10- Performance comparison between caching and non-caching data

These two steps of tuning make a considerable difference in anonymisation performance by MDSBA. Before tuning Spark, it is essential to building a robust Spark cluster and nodes. The hardware infrastructure is the major factor for big data operations success and performance. As mentioned earlier, the memory size of each worker, the number of cores, the number of workers, and the network connection between workers and storages, should be large enough to accommodate the massive size of data. Data anonymisation and analytics are heavy processes that require a powerful and large cluster specifications.

## Summary

With the recent big data’s revolutionary growth, various processing frameworks were developed. Some of these frameworks best fit streaming data, while others can be applied to batch data. In this chapter, two popular frameworks were experimented for *k-anonymity* method. Both of MapReduce and Spark were examined in MDSBA anonymisation process. The aim was paving a reliable ground for MDSBA state-of-the-art anonymisation with the most reliable performance. The experiments show few hurdles in each framework. However, Spark is faster, and more fault tolerant as an in-memory operations framework. In congested networks, Spark reduces data transmission between memory and disks on serializing data with RDD. Spark can be many times faster than MapReduce in anonymisation. To avoid SQL Spark performance degradation, the memory should be larger than the processed data. SQL Spark boosts performance on the large size of memory. Other Spark’s tuning methods that may leverage the anonymisation performance are; UDF algorithm, filter/group commands, and caching data in memory.

On the other side, MapReduce is an old framework that can perform better when memory resources are quite small. This is conditioned by the network traffic and congestion level. MapReduce may operate on small memory resources, but it requires non-busy network, because it relies on a heavy transmission between memory and disks. In the conclusion, It is recommended to implement MDSBA in Spark framework, however, the cluster infrastructure must be prepared well, by providing enough memory and processor resources for each node.

Our future research will focus on finding a proper method for data stream anonymisation. In big data, streaming is an essential type in most data applications. The current anonymisation method of MDSBA does not support the data stream anonymisation. Most recent big data frameworks provide complete solutions for the data stream. We may need to amend the current MDSBA method to cope with the continuous data streaming.

# - CONCLUSION

Analytics in big data is maturing and moving towards mass adoption. The emergence of analytics increases the need for innovative tools and methodologies to protect data against privacy violation. Many data anonymisation methods were proposed to provide some degree of privacy protection by applying data suppression and other distortion techniques. However, currently available methods suffer from poor scalability, performance and lack of framework standardization. Current anonymisation methods are unable to cope with the massive size of data processing. Some of these methods were especially proposed for MapReduce framework to operate in Big Data. However, they still operate in conventional data management approaches. Therefore, there were no remarkable gains in the performance. To address these shortcomings, this thesis proposed a framework that can operate in MapReduce environment to benefit from its advantages, as well as from those in Hadoop ecosystems. The framework provides a granular user’s access that can be tuned to different authorization levels. The proposed solution provides a fine-grained alteration based on the user’s authorization level to access MapReduce domain for analytics. By using well-developed role-based access control approaches, this framework is capable of assigning roles to users and mapping them to relevant data attributes.

The research followed a logical sequence of finding the hinders that faced the current anonymisation methods of big data. The impairments were related to the concept of choosing the best Quasi-Identifier to generalize or specialize attributes. Moreover, the random split, the poor algorithms, the lack of gradual-access, and the limited number of Q-IDs caused a high degradation in anonymisation performance and security. The research has moved from a fact regarding the big data equivalency. The data equivalency increases parallel with the increase number of records. This fact indicates the major difference between traditional data and big data. Generally, big data equivalency is high, specially, in Q-ID attributes that have small number of values. Having these concerns in the current anonymisation methods, and knowing some equivalency facts helps us to outline the shortcomings of the current anonymisation methods. Therefore, Multi-Dimensional Sensitivity-Based Anonymisation methods (MDSBA) was developed to overcome the current impairments and to harness the big data anonymisation operations.

MDSBA core concept was derived from probability and Q-ID aggregation. The probability concept has expedited intensive computation. The current anonymisation algorithm iterates several times to determine the best generalized or specialized Q-ID. In MDSBA, the best generalized Q-ID is determined prior the anonymisation operation. The lowest Q-ID probability value is intuitively generalized. Therefore, there is no need to calculate the best score for each Q-ID attribute. The anonymisation process does not impose a direct iteration, instead, the grouping action is repeated several times for generalisation. The process starts by grouping all Q-IDs first to filter out the fully equivalent records. The remained records are not fully equivalent. Therefore, all Q-IDs are grouped except the one with the lowest Q-ID probability. The lowest one is generalized by the interval masking or the taxonomy tree. Once again, all Q-IDs are grouped except the lowest two Q-IDs probabilities. Similarly, the lowest two Q-IDs are generalized by the interval masking or the taxonomy tree. This operation of grouping continues till the grouping aggregates only one Q-ID. The masking is finally applied to all Q-IDs except the one with the highest probability. The few left over non-equivalent groups are totally suppressed. The equivalency is measured by the ownership level *k̄* instead of *k*. this is important to keep the *k* value constant, while the value of *k̄* increases graduallyas per user’s access.

The current anonymisation methods split data randomly to fit in the limited memory size. This random split reduces the gained information and increases the number of non-equivalent records. For this reason, MDSBA splits data logically as per class value. Sensitive class consists of limited number of values, and each value is aggregated in one group and processed separately. This step is initially conducted before the aggregation and anonymisation start. The number of aggregated groups equals to the number of class values. This logical split is essential to avoid data overflow to the memory. The parallel distributed framework is able to manage a large data size to a certain extent, since the very large data size may increase error rates and pitfalls, which may unexpectedly terminate the process. The second core concept of MDSBA is the aggregation of Q-IDs. Every two to four Q-ID groups are horizontally aggregated and mapped to one or more of the business roles. Multi-dimensional data requires a large number of Q-IDs, which may exceed tens or even more. The current anonymisation methods accept a limited number of Q-IDs, which may not exceed eight or nine. The more Q-IDs added will reduce the performance and more computation time is needed. However, the recent decade has witnessed a technology revolution in social media, which enabled adversaries to develop new attacking scenarios and techniques. This recalled a need to increase the number of Q-IDs, with keeping the processing costs low. MDBSA participated in increasing the number of Q-IDs, and keeping a low processing time. This was structured by aggregating the Q-IDs into small groups.

The thesis starts proposing MDSBA in chapter three. The chapter introduces a preliminarily definition of probabilities and aggregations of MDSBA. The research established some mathematical equations aiming to providing a gradual level of anonymisation according to user’s access level. The equations find the sensitivity level for users. Thus, users with large sensitivity levels may gain less information. Moreover, the sensitivity level can be affected by the time factor. Since datasets importance degrades with the time, which causes a sensitivity level decrease as a result.

After introducing MDSBA, the research sequence experimented MDSBA algorithms in a real big data framework. The experiments examined MDSBA anonymisation results, and its impact on prediction results. The classification error was used as a benchmark to determine the amount of information loss. The classification error was compared with the other anonymisation methods in BUG and TDS. MDSBA failed with the small data size, and succeeded with the larger data size. BUG and TDS showed less classification error rate on anonymising small datasets, while MDSBA showed less classification error rate on anonymising large datasets. Measuring the classification error rate is not an accurate method of measuring the information loss, since different classification methods may output various results. Thus, another benchmark was proposed to measure the actual information loss rather than measuring the prediction level. The benchmark, known by Disruption, showed lower disruption results on large size of datasets. The disruption is even lower when data is split into smaller data files, or on using a larger value of *k*. Moreover, MDSBA computation cost is much lower than BUG and TDS methods in big datasets.

To accomplish one of the main objectives of this research in finding a solution to big data gradual anonymisation, a complete framework is proposed. The framework is distributed between the federation service and the service provider. The framework comprises three various services: core, initialiser, and anonymiser. The core service is embedded in the federation service, which converts user’s requests to assertions transmitted over SAML service. The service provider stores the requested datasets in parallel with the initialiser and anonymiser services. The initialiser service operates on the edge server of the service provider, which generates the anonymisation script, and transfers the script and other contextual files to the MapReduce domain. The anonymiser service completes the task by conducting the anonymisation process. The anonymised copy of data will be created and ready for the user’s access in a secure enclosed directory. The anonymised copy can be generated from production (live) or archived data. The archived data create a stable anonymised copy, which can be available for users as long as the obsolescence value did not expire. On the other hand, the production data expiry may last for few hours or days only.

The research investigated all possible shortcoming in MDSBA security. Two impairments are found in MDSBA structure; Obvious Guess and Across Group Unique Identifiers (AGUI). Several experiments are carried out to eliminate these two security breaches. The experiments results showed that preventing Obvious Guess is possible, by starting an early filtration step. However, AGUI cannot be totally prevented, but it can be mitigated by increasing the *k* value. These conclusions leaded the research to suggest a large number of *k* values. The chosen large *k* value should not affect the gained information negatively. Therefore, Chapter 6 suggested some heuristic-based approaches to find the optimal values of *k*. Some mathematical concepts are applied to find as large *k* value as possible. Cumulative frequency and linear regression are used to move upward of *k* value, and starting from *k*=20. The *k* value increases by 10 until the disparity between the actual value and the linear regression line is high. Moreover, the research suggested some security levels controlled by data owners. The security levels are high, medium, and low. The security parameters are controlled by the number of inhibitors and *k* value interval. The number of inhibitors belongs to the linear regression. Increasing the number of inhibitors will reduce the security level.

Finally, MDSBA was executed in a new parallel framework, known by Spark. It is a memory-base framework, which performs faster than MapReduce. Spark was tested and compared with MapReduce. Two similar algorithms are programmed in Pig Latin script for MapReduce and in Scala script for Spark. The results of the experiments showed a better performance in Spark processing time, when data size fits the memory size. Spark performance has degraded on increasing the data size, while keeping a fixed memory size. In contrast, MapReduce performed better on increasing the data size. MapReduce operates well with the limited resource, and unlike Spark which requires larger resources.

This thesis has answered the research questions by demonstrating that it is possible to provide the data owners with methods that enable them to control the granularity of the user’s access in big data analytics. The research has further demonstrated that it is possible to implement a framework that is able apply an access control model. The model can enforce the organizational business roles over big data. The framework is able to increase the analytics performance and reduce the information loss, which may occur as a symptom of data anonymisation. MDSBA method has made some significant improvement for data obfuscation in security and performance approaches. It has proven that applying such a method is desirable in big data. However, MDSBA suffers some hiccups regarding multiple G(QID) groups, and the possibility of some records re-identification. Across Groups Unique Identifier (AGUI) may appear in some cases. Even with low chances of re-identification, AGUI remains the major hurdle in MDSBA framework. Moreover, eliminating AGUI from data records may create a hitch in MDSBA performance.

It should be noted that the experiments in this thesis were conducted in the university lab. The lab is established by virtual machines infrastructure, which was not fully qualified for MapReduce or Spark domains. Also, it contained limited resources of name nodes and workers or data nodes. Experiments were conducted on a maximum of four virtual machines with limited memory and processors. Such humble infrastructure and small scale cannot process really large datasets, say in terabytes, as a reason of resources limitation and the congested transmission medium between nodes. During the experiments, the university network was congested most of the time. The slow transmission has clearly appeared on applying Pig Latin script to read/write to/from the disks. For these reasons, gigabytes of data sets are used instead of terabytes. Moreover, many experiments were repeated several times, to avoid the arbitrary tasks failure. The failure was explained in Chapter 7.

The future of data anonymisation will be directed toward finding more advanced method for real-time data. Big data access demands increase with data size increase and the pervasive of data over the cloud. This high demand imposes needs to develop better anonymisation methods that are able to obfuscate data in a real-time or near real-time. This is essential to provide a fast and reliable data for various applications. MDSBA may support the real-time anonymisation in the future, especially with the fast development growth of processing tool.

# References

[1] J. M. Tien, "The Sputnik of servgoods: Autonomous vehicles," *Journal of Systems Science and Systems Engineering,* pp. 1-30, 2017.

[2] J. K. Jinzhu Kong, "A Practical Approach to Improve the Data Privacy of Virtual Machines," ed, 2010, pp. 936-941.

[3] M. a. Alam, *Oracle NoSQL database : real-time big data management for the enterprise*: New York : McGraw-Hill Education, 2014, 2014.

[4] Y. Demchenko, P. Grosso, C. de Laat, and P. Membrey, "Addressing big data issues in Scientific Data Infrastructure," ed, 2013, pp. 48-55.

[5] V. Mayer-Schönberger and K. Cukier, *Big data: A revolution that will transform how we live, work, and think*: Houghton Mifflin Harcourt, 2013.

[6] C. Dwork, "Differential Privacy," *Microsoft,* 2010.

[7] V. Mayer-Schönberger, *Big data a revolution that will transform how we live, work, and think*. Boston: Boston : Houghton Mifflin Harcourt, 2013.

[8] D. Bholat, "Big data and central banks," *Big Data & Society,* vol. 2, p. 2053951715579469, 2015.

[9] H. R. Varian, "Beyond big data," *Business Economics,* vol. 49, pp. 27-31, 2014.

[10] A. R. Cynthia Dwork, "The Algorithmic Foundations of Differential Privacy," *Theoretical Computer Science,* vol. 9, 2014.

[11] P. Mohan, A. Thakurta, E. Shi, D. Song, and D. Culler, "GUPT: privacy preserving data analysis made easy," in *SIGMOD '12*, K. S. Candan, uk, Y. Chen, R. Snodgrass, L. Gravano, and A. Fuxman, Eds., ed: ACM, 2012, pp. 349-360.

[12] A. Roth and T. Roughgarden, "Interactive privacy via the median mechanism," ed, 2010, pp. 765-774.

[13] L. Sweeney, "ACHIEVING -ANONYMITY PRIVACY PROTECTION USING GENERALIZATION AND SUPPRESSION," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems,* vol. 10, pp. 571-588, 2002.

[14] A. Machanavajjhala, D. Kifer, J. Abowd, J. Gehrke, and L. Vilhuber, "Privacy: Theory meets Practice on the Map," ed, 2008, pp. 277-286.

[15] B. C. M. Fung, K. Wang, and P. S. Yu, "Anonymizing Classification Data for Privacy Preservation," *Knowledge and Data Engineering, IEEE Transactions on,* vol. 19, 2007.

[16] M. Ogburn, C. Turner, and P. Dahal, "Homomorphic Encryption," *Procedia Computer Science,* vol. 20, pp. 502-509, 2013.

[17] L. Sweeney, "ANONYMITY: A MODEL FOR PROTECTING PRIVACY " *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems,* vol. 10, pp. 557-570, 2002.

[18] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam, "L-diversity: privacy beyond k-anonymity," ed, 2006, pp. 24-24.

[19] K. Lefevre, D. Dewitt, and R. Ramakrishnan, "Incognito: efficient full-domain K-anonymity," ed, 2005, pp. 49-60.

[20] N. Mohammed, B. Fung, and M. Debbabi, "Walking in the crowd: anonymizing trajectory data for pattern analysis," ed, 2009, pp. 1441-1444.

[21] R. C.-W. Wong, J. Li, A. W.-C. Fu, and K. Wang, "(α, k)-anonymity: an enhanced k-anonymity model for privacy preserving data publishing," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2006, pp. 754-759.

[22] H. Samet, *Multidimensional and Metric Data Structure*: Morgan Kaufmann, 2006.

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

[24] 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.

[25] X. Zhang, C. Liu, C. Yang, J. Chen, S. Nepal, and W. Dou, "A hybrid approach for scalable sub-tree anonymization over big data using MapReduce on cloud," vol. 80, ed, 2014, pp. 1008-1020.

[26] 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.

[27] K. D. L. a. E. Lewis, "Web Single Sign-On Authentication using SAML," 2009.

[28] V. Govindaraju, *Big Data Analytics*: Oxford : Elsevier Science, 2015., 2015.

[29] S. Weil, S. Brandt, E. Miller, D. Long, and C. Maltzahn, "Ceph: a scalable, high-performance distributed file system," ed, 2006, pp. 307-320.

[30] H. Hu, Y. Wen, T.-S. Chua, and X. Li, "Toward scalable systems for big data analytics: A technology tutorial," *IEEE access,* vol. 2, pp. 652-687, 2014.

[31] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *Proceedings of the 1st ACM symposium on Cloud computing*, 2010, pp. 143-154.

[32] K. Keeton, C. B. Morrey III, C. A. Soules, and A. Veitch, "Lazybase: freshness vs. performance in information management," *ACM SIGOPS Operating Systems Review,* vol. 44, pp. 15-19, 2010.

[33] K. Chodorow, *MongoDB*. Sebastopol: Sebastopol : O'Reilly Media, 2010.

[34] J. Murty, *Programming amazon web services: S3, EC2, SQS, FPS, and SimpleDB*: " O'Reilly Media, Inc.", 2008.

[35] J. P. Ming Hua, *Privacy-Preserving Data Mining: Models and Algorithms*. Boca Raton, FL: Springer, 2008.

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

[37] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," *HotCloud,* vol. 10, p. 95, 2010.

[38] J. R. Evans and C. H. Lindner, "Business analytics: the next frontier for decision sciences," *Decision Line,* vol. 43, pp. 4-6, 2012.

[39] J. Richardson, K. Schlegel, B. Hostmann, and N. McMurchy, "Magic quadrant for business intelligence platforms," *Core research note G,* vol. 154227, 2008.

[40] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Fog computing: A platform for internet of things and analytics," in *Big Data and Internet of Things: A Roadmap for Smart Environments*, ed: Springer, 2014, pp. 169-186.

[41] C. Doherty, M. Lang, J. Deane, and R. Connor, *Information disclosure on social networking sites: An exploratory survey of factors impacting user behaviour on facebook*, 2014.

[42] X. Wu, X. Zhu, G. Q. Wu, and W. Ding, "Data mining with big data," *IEEE Trans Knowl Data Eng,* vol. 26, 2014.

[43] L. S. J. Z. J. S. Y. S. Hooley. (2016). *The data map*. Available: <https://thedatamap.org/map2013/p17.php>

[44] S. Amini, I. Gerostathopoulos, and C. Prehofer, "Big data analytics architecture for real-time traffic control," in *Models and Technologies for Intelligent Transportation Systems (MT-ITS), 2017 5th IEEE International Conference on*, 2017, pp. 710-715.

[45] A. Gandomi and M. Haider, "Beyond the hype: Big data concepts, methods, and analytics," *International Journal of Information Management,* vol. 35, pp. 137-144, 2015.

[46] J. Gantz and D. Reinsel, "The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east," *IDC iView: IDC Analyze the future,* vol. 2007, pp. 1-16, 2012.

[47] A. Bifet, "Mining Big Data in Real Time," *Informatica,* vol. 37, pp. 15-20, 2013.

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

[49] H. Project. (2016). *HDFS Users Guide*. Available: <https://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html>

[50] Hortonworks. (2016). *Determine YARN and MapReduce Memory Configuration Settings*. Available: <https://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.0.6.0/bk_installing_manually_book/content/rpm-chap1-11.html>

[51] A. F. Gates, O. Natkovich, S. Chopra, P. Kamath, S. M. Narayanamurthy, C. Olston*, et al.*, "Building a high-level dataflow system on top of Map-Reduce: the Pig experience," *Proceedings of the VLDB Endowment,* vol. 2, pp. 1414-1425, 2009.

[52] Y. Gahi, M. Guennoun, and H. T. Mouftah, "Big Data Analytics: Security and privacy challenges," vol. 2016-, ed, 2016, pp. 952-957.

[53] A. Haeberlen, B. C. Pierce, and A. Narayan, "Differential Privacy Under Fire," in *USENIX Security Symposium*, 2011.

[54] C. Dwork, G. N. Rothblum, and S. Vadhan, "Boosting and differential privacy," in *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, 2010, pp. 51-60.

[55] J. Tang, A. Korolova, X. Bai, X. Wang, and X. Wang, "Privacy Loss in Apple's Implementation of Differential Privacy on macOS 10.12," *arXiv preprint arXiv:1709.02753,* 2017.

[56] S. T. S. Indrajit Roy, Ann Kilzer, Vitaly Shmatikov, Emmett Witchel, "Airavat: Security and Privacy for MapReduce," *The University of Texas at Austin,* 2010.

[57] M. Nassar, A. A.-R. Orabi, M. Doha, and B. Al Bouna, "An SQL-like query tool for data anonymization and outsourcing," in *Cyber Situational Awareness, Data Analytics and Assessment (CyberSA), 2015 International Conference on*, 2015, pp. 1-3.

[58] Z. C. Schreuders, T. J. McGill, and C. Payne, "Towards usable application-oriented access controls: qualitative results from a usability study of SELinux, AppArmor and FBAC-LSM," *International Journal of Information Security and Privacy,* vol. 6, pp. 57-76, 2012.

[59] N. Li, W. Qardaji, and D. Su, "On sampling, anonymization, and differential privacy or, k-anonymization meets differential privacy," in *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, 2012, pp. 32-33.

[60] L. Sweeney, "Datafly: A system for providing anonymity in medical data," in *Database Security XI*, ed: Springer, 1998, pp. 356-381.

[61] R. Motwani and Y. Xu, "Efficient algorithms for masking and finding quasi-identifiers," in *Proceedings of the Conference on Very Large Data Bases (VLDB)*, 2007, pp. 83-93.

[62] A. Meyerson and R. Williams, "On the complexity of optimal K-anonymity," in *PODS '04*, C. Beeri, Ed., ed: ACM, 2004, pp. 223-228.

[63] R. J. Bayardo and R. J. Rakesh Agrawal, "Data privacy through optimal k-anonymization," ed. USA, 2005, pp. 217-228.

[64] A. Scott, V. Srinivasan, and U. Stege, "k-Attribute-Anonymity is hard even for k= 2," *Information Processing Letters,* vol. 115, pp. 368-370, 2015.

[65] F. Y. Kuo and I. H. Sloan, "Lifting the curse of dimensionality," *Notices of the AMS,* vol. 52, pp. 1320-1328, 2005.

[66] S. Agrawal and J. R. Haritsa, "A framework for high-accuracy privacy-preserving mining," in *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, 2005, pp. 193-204.

[67] F. Li, J. Sun, S. Papadimitriou, G. A. Mihaila, and I. Stanoi, "Hiding in the crowd: Privacy preservation on evolving streams through correlation tracking," in *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, 2007, pp. 686-695.

[68] C. C. Aggarwal, "On k-anonymity and the curse of dimensionality," in *Proceedings of the 31st international conference on Very large data bases*, 2005, pp. 901-909.

[69] C. C. Aggarwal, "Privacy and the dimensionality curse," *Privacy-Preserving Data Mining,* pp. 433-460, 2008.

[70] H. Peng, F. Long, and C. Ding, "Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy," *IEEE Transactions on pattern analysis and machine intelligence,* vol. 27, pp. 1226-1238, 2005.

[71] H. Zakerzadeh, C. Aggarwal, and K. Barker, "Managing dimensionality in data privacy anonymization," *An International Journal,* vol. 49, pp. 341-373, 2016.

[72] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam, "L-diversity: Privacy beyond k-anonymity," *ACM Transactions on Knowledge Discovery from Data (TKDD),* vol. 1, pp. 3-es, 2007.

[73] S. Ninghui Li, S. Tiancheng Li, and S. Venkatasubramanian, "t-Closeness: Privacy Beyond k-Anonymity and l-Diversity," ed, 2007, pp. 106-115.

[74] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam, "l-diversity: Privacy beyond k-anonymity," in *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*, 2006, pp. 24-24.

[75] N. Mohammed, B. C. M. Fung, P. C. K. Hung, and C.-k. Lee, "Centralized and Distributed Anonymization for High-Dimensional Healthcare Data," *ACM Transactions on Knowledge Discovery from Data (TKDD),* vol. 4, pp. 1-33, 2010.

[76] J. Han, J. Yu, H.-q. Yu, and J. Jia, "Individuation privacy preservation oriented to sensitive values," *Acta Electronica Sinica,* vol. 38, pp. 1723-1728, 2010.

[77] J. Pardalos and M. Resende, "On maximum clique problems in very large graphs," *DIMACS series,* vol. 50, pp. 119-130, 1999.

[78] A. IRUDAYASAMY and L. Arockiam, "SCALABLE MULTIDIMENSIONAL ANONYMIZATION ALGORITHM OVER BIG DATA USING MAP REDUCE ON PUBLIC CLOUD," *Journal of Theoretical & Applied Information Technology,* vol. 74, 2015.

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

[80] M. Balusamy and S. Muthusundari, "Data anonymization through generalization using map reduce on cloud," in *Computer Communication and Systems, 2014 International Conference on*, 2014, pp. 039-042.

[81] B. B. Mehta and U. P. Rao, "Privacy preserving big data publishing: a scalable k-anonymization approach using MapReduce," *IET Software,* vol. 11, pp. 271-276, 2017.

[82] X. Zhang, L. T. Yang, C. Liu, and J. Chen, "A scalable two-phase top-down specialization approach for data anonymization using MapReduce on cloud," *IEEE Trans Parallel Distrib Syst,* vol. 25, 2014.

[83] P. Russom, "BIG DATA ANALYTICS," *TDWI RESEARCH/IBM,* vol. 4, 2011.

[84] P. Samarati, "Protecting respondents identities in microdata release," *Knowledge and Data Engineering, IEEE Transactions on,* vol. 13, pp. 1010-1027, 2001.

[85] N. Qing Zhang, D. Koudas, D. Srivastava, and D. Ting Yu, "Aggregate Query Answering on Anonymized Tables," ed, 2007, pp. 116-125.

[86] K. G. Shin, X. Ju, Z. Chen, and X. Hu, "Privacy protection for users of location-based services," *IEEE Wireless Communications,* vol. 19, pp. 30-39, 2012.

[87] Y. Kwon, "Hadoop Internals," 2014.

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

[89] MatLab. (2014). *MatLab and SimuLink*. Available: <https://au.mathworks.com>

[90] S. G. Shui Yu, *Big Data Concepts, Theories, and Applications*: Springer, 2016.

[91] C. Ji, Y. Li, W. Qiu, U. Awada, and K. Li, "Big data processing in cloud computing environments," in *Pervasive Systems, Algorithms and Networks (ISPAN), 2012 12th International Symposium on*, 2012, pp. 17-23.

[92] H. Chen, R. H. Chiang, and V. C. Storey, "Business intelligence and analytics: From big data to big impact," *MIS quarterly,* vol. 36, 2012.

[93] A. Holmes, *Hadoop in practice*. Shelter Island, NY: Shelter Island, NY : Manning, 2012.

[94] D. B. Kirk and W. H. Wen-Mei, *Programming massively parallel processors: a hands-on approach*: Morgan kaufmann, 2016.

[95] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: portable parallel programming with the message-passing interface* vol. 1: MIT press, 1999.

[96] A. Lesuisse, *Data processing with Pig Latin*: Année académique 2011.

[97] 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.

[98] M. Al-Zobbi, S. Shahrestani, and C. Ruan, "Sensitivity-based anonymization of big data," in *Local Computer Networks Workshops (LCN Workshops), 2016 IEEE 41st Conference on*, 2016, pp. 58-64.

[99] K. W. Benjamin C.M. Fung, Ada Wai-Chee Fu , Philip S. Yu, *Introduction to Privacy-Preserving Data Publishing: Concepts and Techniques*: CRC Press, 2011.

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

[101] 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.

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

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

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

[104] 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.

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

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

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

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

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

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

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

[112] 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.

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

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

[115] T. Dokeroglu, S. Ozal, M. A. Bayir, M. S. Cinar, and A. Cosar, "Improving the performance of Hadoop Hive by sharing scan and computation tasks," *Journal of Cloud Computing,* vol. 3, p. 1, 2014.

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

[117] 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.

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

[119] B. Lublinsky, *Professional Hadoop Solutions*. Hoboken: Hoboken : Wiley, 2013.

[120] A. Gates and D. Dai, *Programming Pig: Dataflow Scripting with Hadoop*: " O'Reilly Media, Inc.", 2016.

[121] G. a. R. Peter. (2010, 23 Jan 2018). *Pig*. Available: <https://www2.cs.duke.edu/courses/cps296.1/spring10/lectures/21-pig.pdf>

[122] G. P. Gibilisco, M. Li, L. Zhang, and D. Ardagna, "Stage aware performance modeling of DAG based in memory analytic platforms," in *Cloud Computing (CLOUD), 2016 IEEE 9th International Conference on*, 2016, pp. 188-195.

[123] K. Elmeleegy, C. Olston, and B. Reed, "Spongefiles: Mitigating data skew in mapreduce using distributed memory," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, 2014, pp. 551-562.

[124] V. Kalavri, V. Vlassov, and P. Brand, "Ponic: Using stratosphere to speed up pig analytics," in *European Conference on Parallel Processing*, 2013, pp. 279-290.

[125] 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.

[126] 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.

[127] P. S. Ke Wang, S. Yu, and S. Chakraborty. (2004) Bottom-up generalization: a data mining solution to privacy protection. 249-256.

[128] 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.

[129] O. Org, "Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0," p. 46, 15 March 200 2005.

[130] Venkadesh.M and A. C. S. Dr, "Study on Improving Web Security using SAML Token," *International Journal of Computer Technology and Applications,* vol. 04, p. 917, 2013.

[131] J. Paschoud, "SHIBBOLETH AND SAML: AT LAST, A VIABLE GLOBAL STANDARD FOR RESOURCE ACCESS MANAGEMENT," *New Review of Information Networking,* vol. 10, pp. 147-160, 2004.

[132] C. M. Garcia and R. Abilio, "Systems Integration Using Web Services, REST and SOAP: A Practical Report," *Revista de Sistemas de Informação da FSMA,* pp. 34-41, 2017.

[133] O. Canovas, G. Lopez, and A. F. Gomez-Skarmeta, "A credential conversion service for SAML-based scenarios," *PUBLIC KEY INFRASTRUCTURE, PROCEEDINGS,* vol. 3093, pp. 297-305, 2004.

[134] J. B. Rosenberg, *Securing Web services with WS-Security : demystifying WS-Security, WS-Policy, SAML, XML Signature, and XML Encryption*. Indianapolis, IN: Indianapolis, IN : SAMS, 2004.

[135] G. S. coding. (2015). *phpseclib: SSH2 Feature List*. Available: <http://phpseclib.sourceforge.net/ssh/intro.html>

[136] R. A. Napier, "Secure automation: achieving least privilege with SSH, Sudo and Setuid," in *18th Large Installation System Administration Conference*, 2004, pp. 203-212.

[137] Y. X. Rajeev Motwani, "Efficient Algorithms for Masking and Finding Quasi-Identifiers," 2007.

[138] G. Server. (2016). *Installation Guide for Gluu Server*. Available: <https://gluu.org/docs/installation-guide/install/>

[139] N. C. Institute. (2013). *Accessing the 1973-2013 SEER Data*. Available: <http://seer.cancer.gov/data/access.html>

[140] IBM. (2016). *IBM SPSS Software*. Available: <https://www.ibm.com/analytics/us/en/technology/spss/>

[141] J.-W. Byun, A. Kamra, E. Bertino, and N. Li, "Efficient k-anonymization using clustering techniques," in *International Conference on Database Systems for Advanced Applications*, 2007, pp. 188-200.

[142] M. Al-Zobbi, S. Shahrestani, and C. Ruan, "Multi-dimensional sensitivity-based anonymization method for big data," in *Networks of the future: architectures, technologies, and implementations*, M. Elkhodr, S. Shahrestani, and Q. Hassan, Eds., ed Boca Raton: Francis, 2017.

[143] T. A. Su and G. Ozsoyoglu, "Controlling FD and MVD Inferences in Multilevel Relational Database Systems," *IEEE Transactions on Knowledge and Data Engineering,* vol. 3, pp. 474-485, 1991.

[144] M. Morgenstern, *Security and inference in multilevel database and knowledge-base systems* vol. 16: ACM, 1987.

[145] M. Smith, C. Szongott, B. Henne, and G. Von Voigt, "Big data privacy issues in public social media," in *Digital Ecosystems Technologies (DEST), 2012 6th IEEE International Conference on*, 2012, pp. 1-6.

[146] S. Yu, "Big privacy: Challenges and opportunities of privacy study in the age of big data," *IEEE access,* vol. 4, pp. 2751-2763, 2016.

[147] R. Lu, H. Zhu, X. Liu, J. Liu, and J. Shao, "Toward efficient and privacy-preserving computing in big data era," *IEEE Network,* vol. 28, pp. 46-50, 2014.

[148] J. P. Daries, J. Reich, J. Waldo, E. M. Young, J. Whittinghill, A. D. Ho*, et al.*, "Privacy, anonymity, and big data in the social sciences," *Communications of the ACM,* vol. 57, pp. 56-63, 2014.

[149] E. Kabir, A. Mahmood, H. Wang, and A. Mustafa, "Microaggregation sorting framework for k-anonymity statistical disclosure control in cloud computing," *IEEE Transactions on Cloud Computing,* 2015.

[150] H. Park and K. Shim, "Approximate algorithms for k-anonymity," in *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, 2007, pp. 67-78.

[151] R. Hariharan, C. Mahesh, P. Prasenna, and R. V. Kumar, "Enhancing privacy preservation in data mining using cluster based greedy method in hierarchical approach," *Indian Journal of Science and Technology,* vol. 9, 2016.

[152] A. Basu, T. Nakamura, S. Hidano, and S. Kiyomoto, "k-anonymity: Risks and the Reality," in *Trustcom/BigDataSE/ISPA, 2015 IEEE*, 2015, pp. 983-989.

[153] B. I. Susan Dean. (2010, 29/12/2016). *Sampling and Data: Frequency, Relative Frequency, and Cumulative Frequency*. Available: <https://www.saylor.org/site/wp-content/uploads/2011/06/MA121-1.1.3-3rd.pdf>

[154] N. R. Draper and H. Smith, *Applied regression analysis*: John Wiley & Sons, 2014.

[155] S. Chatterjee and A. S. Hadi, *Regression analysis by example*: John Wiley & Sons, 2015.

[156] K. G. Steven Ruggles, Ronald Goeken, Josiah Grover, and Matthew Sobek, "Integrated Public Use Microdata Series: Version 6.0 [dataset]. Minneapolis," in *MN: University of Minnesota*, M. U. o. Minnesota, Ed., 6.0 ed. [www.ipums.org:](http://www.ipums.org:) IPUMS-USA, 2015.

[157] S. Gopalani and R. Arora, "Comparing apache spark and map reduce with performance analysis using K-means," *International Journal of Computer Applications,* vol. 113, 2015.

[158] J. Shi, Y. Qiu, U. F. Minhas, L. Jiao, C. Wang, B. Reinwald*, et al.*, *Clash of the titans: Mapreduce vs. spark for large scale data analytics* vol. 8, 2015.

[159] A. G. Shoro and T. R. Soomro, "Big data analysis: Apache spark perspective," *Global Journal of Computer Science and Technology,* vol. 15, 2015.

[160] M. Frampton, *Mastering Apache Spark*: Packt Publishing Ltd, 2015.

[161] M. A. Abbasi, *Learning Apache Spark 2*, 1st ed. ed.: Birmingham : Packt Publishing, 2017., 2017.

[162] M. Li, J. Tan, Y. Wang, L. Zhang, and V. Salapura, "Sparkbench: a comprehensive benchmarking suite for in memory data analytic platform spark," in *Proceedings of the 12th ACM International Conference on Computing Frontiers*, 2015, p. 53.

[163] D. B. West, *Introduction to graph theory* vol. 2: Prentice hall Upper Saddle River, 2001.

[164] Y. Low, J. E. Gonzalez, A. Kyrola, D. Bickson, C. E. Guestrin, and J. Hellerstein, "Graphlab: A new framework for parallel machine learning," *arXiv preprint arXiv:1408.2041,* 2014.

[165] R. C. Ionescu, "A scalable system for primal-dual optimization," *arXiv preprint arXiv:1507.01456,* 2015.

[166] J. Lee, S. Cozens, and P. Wainwright, "Object-Oriented Perl," in *Beginning Perl*, ed: Springer, 2004, pp. 253-285.

[167] D. Chen and H. Zhao, "Data security and privacy protection issues in cloud computing," in *Computer Science and Electronics Engineering (ICCSEE), 2012 International Conference on*, 2012, pp. 647-651.

[168] M. Zaharia, T. Das, H. Li, S. Shenker, and I. Stoica, "Discretized Streams: An Efficient and Fault-Tolerant Model for Stream Processing on Large Clusters," *HotCloud,* vol. 12, pp. 10-10, 2012.

[169] M. Al-Zobbi, S. Shahrestani, and C. Ruan, "Implementing A Framework for Big Data Anonymity and Analytics Access Control," in *Trustcom/BigDataSE/ICESS, 2017 IEEE*, 2017, pp. 873-880.