In-Situ Anonymization of Big Data

Tomislav Križan, Marko Brakus, Davorin Vukelić
Poslovna Inteligencija d.o.o., Zagreb, Croatia
{tomislav.krizan, marko.brakus, davorin.vukelic}@inteligencija.com

Abstract - With organizations storing and even openly publishing their data for further processing, privacy becomes an issue. Such open data should retain its original structure while protecting sensitive personal data. Our aim was to develop fast and secure software for offline anonymization of (distributed) big data. Herein, we describe speed and security requirements for anonymization systems, popular techniques of anonymization and de-anonymization attacks. We give a detailed description of our software for in-situ anonymization of big data distributed in a cluster tested on a real Teleco customer data record (CDR) dataset (dataset size is around 500 GB).

I. INTRODUCTION

Vast amount of data is being generated from versatile sources and organizations are primarily interested in analyzing it for further information and improved decision making. Data analysis surely leverages data value for the organization and opening data publicly benefits wider community and fosters further scientific research. However, analysis means disclosing data to other employees, third party organizations or even publishing it on the internet for everybody to use. Stored data is often secret, proprietary or protected by data protection and privacy laws and organization can suffer significant financial loss if that data is misused. Huge cost of data theft and misuse urged organizations to improve security measures and protect data.

Ideally, data would be void of any sensitive information which is protected by law while retaining all other information and structure valuable for further analysis. These sensitive parts of dataset need to be changed in a way to make it impossible or at least infeasible to trace the changed values back to the original ones. Such a transformation would ensure that confidential data, even when outsourced, published or stolen, could not be misused. Anonymization (de-identification, depersonalization, masking, obfuscation, etc.) is a process that changes or removes sensitive data while preserving its structure. It is considered successful if it is impossible or computationally infeasible to track original sensitive data from the anonymized dataset.

Anonymization is traditionally done on organizations’ files, databases or centralized warehouses. Data is usually extracted from different sources, validated, transformed and loaded into databases. Due to nonstandard data demands, organizations employ newer Big Data technologies characterized by increased data volume (great data size), velocity (real-time and near-real-time systems to meet high-speed data streams) and variety (lack of data structure). Such high volume of data should be processed preferably at velocity close to the one at which it was generated so that it can possibly be included in a data pipeline. Distributed environments and traditional centralized processing assume costly data movements (CPU time, network bandwidth and energy consumption) and additional vertical scaling on the place of processing. Therefore, data should preferably be processed at the same place where it was generated to avoid costly data movement. In-situ methods are inspired by such demands, where inter-node communication should be reduced. Thus, data movement should be minimized by doing computing locally where data was created. One of the most common applications of the in-situ processing is in visualization by doing rendering on the nodes where data was generated [1]. Also, in-situ method is appropriate for monitoring distributed environment [2] generating real time streams of data (network equipment, sensors, etc.).

In this work, we give a description of anonymization process which encompasses defining organizations’ sensitive data, setting requirements for anonymization system, choosing appropriate anonymization techniques and considering their vulnerabilities. We then present our implemented solution for anonymization of Big Data inspired by in-situ data processing. It is exhibited how various aspects of Big Data are met with in-memory object store for high data volumes and fast messaging for inter-node communication. It is shown how the system is designed for anonymization of data at the nodes where data is
created by minimum client-server communication. Lastly, we present system performance measurements for anonymizing an example telco mobile customer data record (CDR) dataset in a distributed environment.

A. Types of Sensitive Information

Personal data [3] is defined as “any information relating to an identified or identifiable natural person ("data subject"); an identifiable person is one who can be identified, directly or indirectly, in particular by reference to an identification number or to one or more factors specific to his physical, physiological, mental, economic, cultural or social identity.” Some examples of such data are: address, credit card number, bank statements, criminal record.

Sensitive data attributes/fields/columns/identifiers can be classified into several types regarding their sensitivity. Direct (explicit) identifiers contain values that can uniquely identify persons or organizations (i.e. sensitive entities). Examples are name, telephone number, email address, credit card number, various IDs, etc. Indirect (implicit, quasi) identifiers can identify the entity but only indirectly. Examples include geographical location, gender, marital status, profession, date of birth, etc.

B. Data Anonymization System Requirements

Data anonymization is meant to make restricted and confidential data harder to misuse and available to unauthorized people, both inside and outside of the organization.

Even (database column-level) encryption using SQL scripts is a valid approach for data anonymization. If the databases are different, inconsistency may pose a problem, i.e. the same customer would appear differently in each database. There could also be inconsistency between departments using different versions of the same technology and “referential integrity” thus cannot be maintained. Also, such scripts are not flexible and hard to maintain. Therefore, a specialized data masking tool would be more appropriate for any enterprise-wide data anonymization initiative.

A system for anonymization should adhere to several requirements, primarily security and speed.

Security should be strong enough to make any tracing of masked (anonymized) values back to original sensitive information hard enough or impossible. This can not only be achieved by strong encryption, where used keys also need to be protected properly, but also by other masking methods. Letting only authenticated users run the masking would also improve system’s security.

Speed is measured in number of records masked in unit of time. This capability determine whether the system is capable of working online (real-time mode, data in motion) or only offline (batch mode, data at rest). The process should in any case be finished inside reasonable time.

Techniques. Supported algorithms should be able to anonymize/generate/format all the different value types of the dataset (credit card numbers, zip codes, telephone numbers, etc.). Also, different ways of masking should be supported, e.g. partial masking (masking only part of the value).

Connectivity. With data commonly being stored in databases, the system could offer connecting to various databases, creating queries and preserving physical structure, indices, primary keys, triggers and referential integrity across tables. For input from file, various file formats (fixed width records, delimiters and encodings) should be supported. Output can also be of different types. Integration into a larger data process such as ETL (Extract-Transform-Load) is also desirable.

Resilience. The system should skip/report invalid data such as improper or missing values in a record/row. Checkpointing (marking the last successful record) would enable restarting the masking from before the record of last failure. Also, examining system’s logs should reveal the source of possible failure (improper data or system error).

C. Data Anonymization Techniques

The term anonymity has a different meaning in different countries. In some countries it corresponds to computational anonymity (difficulty to computationally trace original values from the masked ones) and in other countries to perfect anonymity (theoretically impossible to trace original values from masked ones) [4].

There are two main approaches to anonymization: one based on randomization and other on generalization [4]. Randomization comprises any modification of the original values either by removing them (nullifying), replacing them with randomly generated or chosen values, adding noise or swapping. Generalization techniques convert values into more general ones and is therefore not suitable when we want to retain data structure and information.

Techniques can be further described with several characteristics. Deterministic (repeatable) - same original value always maps into the same masked value. Uniqueness – two different original values cannot map into the same masked value. Conditional sensitive - different masking for different given conditions/parameters. Partial – masking only part of the value (e.g. only several last digits of the telephone number). Realistic – masked output looks meaningful. Length preservation - masked value is of the same length as the original. Reversibility – it is possible to get original value from the masked one (e.g. provided the encryption key). Information loss – how much information is lost during the masking.

Substitution technique replaces the original value with some other fake masked value. This fake value can be either randomly chosen from the given list (pseudonymization) of fake values or generated randomly. The original value can also be changed by noise addition where random noise is added to the original value. The variant when the masked value is chosen from the given
list (usually external file) is called lookup substitution. While it replaces original values with meaningful masked fake values its list can easily be exhausted for a large number of different original values.

**Shuffling or data swapping** randomly rearranges values inside one dataset column while preserving the order in other columns. To make masked data more realistic, group shuffling can be used to rearrange several columns together. It is safer to do shuffling on larger dataset where tracing the original values is harder.

**Blurify** technique changes the original value by a given variance. Usually financial data such as salaries are randomly increased or decreased for the specified variance percentage, e.g. \( \pm 5\% \) (Numeric variance). Minimum and maximum limits can be specified along with the variance to keep the masked value within meaningful boundaries. The technique can also be applied to temporal data such as birth dates (Date variance).

**Nulling out** simply deletes the original value. Normally, the technique cannot be used on non-nullable columns of the database.

**Character masking** technique is similar to nulling out and replaces the original value with a specified character constant. E.g. first several digits of a credit card number are masked with a character constant.

**Encryption** technique encrypts the original value into anonymized one by using the provided key. Although the confidentiality is provided, the output value is not realistic. Also, the key needs to be managed and if disclosed, data can be compromised. It is suitable when the anonymized data later needs to be de-anonymized back into original form.

**Hashing** is a form of encryption technique where no key is used and data of arbitrary length is irreversibly masked into fixed length message digest output. Same as with encryption, hashing is wrongly equated with anonymization, and although it may put itself as a first intuitive choice for obfuscating data, it is not advised [4].

**Aggregation** aggregates several clustered records and replaces their values with generalizations. Numerals can be averaged, dates can be generalized into months or years, exact salaries into ranges, etc.

**K-anonymity** is a requirement that pseudo-identifiers of each record must match at least k other records in the anonymized dataset. Optimal k-anonymization is an NP-hard problem and there are various approaches to meet this requirement. L-diversity and t-closeness models are refinements of k-anonymity.

**D. Data De-Anonymization Risks**

Anonymization techniques are as secure as difficult it is to trace the original value from the masked or infer any details about the sensitive information. There is no generally accepted mature method to evaluate this difficulty of de-anonymization [4]. Therefore, choosing the most resistant anonymization technique is nontrivial and dependent on the use case.

According to [4], there are three main approaches to de-anonymization. **Singling out** corresponds to the possibility to isolate record(s) which identify an individual. **Linking attack** tries to link records from different datasets containing common indirect identifiers.

The widespread misbelief about pseudonymization, substitution, encryption or hashing is that if the original sensitive data is replaced by random values or encrypted with state of the art encryption with long enough a key or even completely removed, it is perfectly anonymized [4]. Hashed data can be reversed back to original by brute force attacks or reversing the algorithm. While state of the art encryption does offer high degree of protection, weak custody of the key could make it possible to identify the data. Furthermore, abundant data on the internet can be used to form auxiliary datasets which are then combined with the anonymized ones to help re-identify sensitive information. De-anonymization of Netflix prize is a popular case where data, ostensibly anonymized by randomization, was revealed with the help of auxiliary dataset obtained from the internet [5]. Generally, a dataset with not only perfectly obfuscated but even removed direct identifiers remains vulnerable to singling out and linking to other datasets.

Generalization techniques, while resistant to singling out, require sophisticated tuning against linkability and inference attacks. The main flaw of the k-anonymity is the inference attack, which is improved in l-diversity and t-closeness techniques. Although these refinements are imperfect, they make inference attacks much less successful.

**II. RELATED WORK**

Growing volume of data and privacy concerns spawned work both in newer anonymization techniques and their merging with popular large scale distributed computing systems. These better anonymization techniques strive to be more resilient to background knowledge and linking attacks done by using increasingly available online datasets. Incorporating the anonymization and privacy preservation into distributed frameworks adds special requirements for secured network protocols, prevention of data disclosure to other nodes in the environment and restricting types of data queries.

In [6] a solution for k-anonymizing vertically distributed data is presented. Vertically distributed datasets are first anonymized locally and then joined. A special secured protocol is constructed that achieves global k-anonymization between two separate nodes. [7] describes anonymization of data horizontally distributed among several cluster nodes. It argues that anonymizing data separately on each node fails to meet referential integrity and that centralizing in on a third party secured server is infeasible. Therefore, a special distributed anonymization protocol is implemented that queries distributed data sources and guarantees that such virtual union of datasets is k-anonymous. Similarly, in [8] an
Encrypted global view of the horizontally or vertically distributed dataset is constructed by means of a special secured protocol.

Cloud computing and Big Data offer organizations cost effective computing resources. Yet, organizations are hesitant or legally forbid to use them because of the possible security risks. There are several proposed solutions for privacy protection of Big Data in the cloud. [9] presents data anonymization framework which upgrades MapReduce computing with privacy protection. MapReduce query jobs are constrained by enforced value ranges, preventing unwanted disclosure of original data. The framework enables large-scale computation on data that originates from different sources and belong to different owners. It concentrates on preserving differential privacy by adding noise to the output of the computation. However, data stored in the cloud is not actually anonymized, not all query types are possible and the query results contain added noise. Other two systems described in [10] and [11] provide a privacy-aware cloud computing based on partitioning MapReduce jobs to process sensitive data on private clouds and insensitive data on public clouds. The systems transparently upgrade MapReduce framework and not much adjustments to the existing jobs are needed. It is argued that traditional techniques fail to anonymize large scale data and that it is more efficient to modify or restrict the running MapReduce jobs that access data. Thus, stored data itself remains unanonymized.

III. ANONYMIZATION SYSTEM OVERVIEW

The anonymization system described herein was developed for the purposes of European FP7 FERARI (Flexible Event pRocessing for big dAta aRchitectures) project [12]. The project’s goal is to develop novel in-situ methods for detection of events in distributed environments producing Big Data. The system is originally used to anonymize real world telco Customer Data Records (CDR) dataset, which is then used in the project primarily for the research purposes of developing methods of Deep Packet Investigation (DPI) for Mobile Fraud Detection.

The intention was to develop a generalized Java solution that would work both for data residing on a single node and for data scattered across a distributed environment as exhibited in Figure 1. In addition to anonymization of stationary data, the system attempts to meet demands of high velocity dynamic data. Therefore, multiple inputs are supported, client server communication is minimized and data structures are adjusted for high volume data.

A. Data Connectivity

Connection to standard stationary data repositories such as databases and flat files is supported. Connection to database table is done by JDBC connection and executing the table retrieval query given in the client’s configuration. Currently, two main flat file types are supported: delimited text files and fixed width text files. Description of file type and columns is given in the client’s configuration. Files are read and written by buffering with java.io.BufferedReader and java.io.BufferedWriter, which showed to be the simplest and fastest way of reading and writing text files. Both flat files and table rows retrieved by the query should match the columns description specified in the client’s configuration. Input of data in motion for real time (dynamic) masking is implemented by supporting JMS messaging queue through javax.jms.QueueReceiver. Specifically, ActiveMQ [13] high-performance asynchronous messaging queue library was used for testing. Similarly as for input, data output is provided into database table, delimited or fixed width text file or JMS message queue.

B. Anonymization Techniques

The system currently implements several anonymization techniques of randomizing type. Namely, substitution by pseudonymization, partial substitution, key generator, nullify, blurify and random.

The implemented substitution masking method replaces each value with a fake masked one. The list of possible masked values is given in a file in masker client configuration. Also, the technique can be made deterministic and unique. Similarly, partial substitution masking method replaces only part of the value with a randomly generated characters. The part to be masked is specified by starting position on the left and maximum masked length starting from the right.

The implemented key generator method generates random alphanumeric or only numeric strings of specific length. It can also be made deterministic and unique. Similarly, the random method works on the client side by replacing the value with a random string of specific length and can not be deterministic or unique.

The nullify method simply erases the value by returning an empty string.
To preserve referential integrity among different data sources possibly on different nodes, some mapping data used for anonymization has to be centralized on a server node accessible from all other client nodes. In case of substitution methods, mappings of keys into masked values are kept in a map data structure, which is accessible on the server node.

C. Mapping Data Structure

Pseudonymization masking techniques to be deterministic and preserve referential integrity store mappings of keys into masked values. We implemented hash table of key to masked value mappings for retrieving past mappings. Also, a Bloom filter was implemented for checking if some masked value already exists in the hash table. When processing higher volumes of data, the heap available to Java program can easily become a constraint. The increase of allocated heap also increases the time that the garbage collector needs to collect unused memory objects. Contrary to the popular belief, it is the number of used (live) objects that slows down garbage collection. Garbage collection can therefore greatly and unpredictably increase response time by creating long pauses. Thus, offloading data structures to off-heap native direct memory successfully scales vertically improving performance [14] and adjusting the system for higher volume and higher velocity of Big Data. Access to off-heap direct memory is provided by java.nio.ByteBuffer and java.nio.MappedByteBuffer. Although the former is faster, we employed the later to map the off-heap hash table data structure to a disk file and thus ensure persistence. Such memory mapped file is guaranteed by the OS to be stored to the actual file on disk in the case the program brakes. However, one single java.nio.MappedByteBuffer instance can refer to no more than approximately 2 GB of memory as it is accessed by 32-bit integer index. To overcome this limitation, several overlapping buffers are used to refer to larger memory area. Thus, the complete mapping data structure is located in an array structure inside multiple java.nio.MappedByteBuffer objects pointing to one large memory area.

The map data structure located on the server node is accessed from multiple client nodes (Client Masking Component and Server Masking Component in Figure 1) and is therefore made concurrent. Concurrency is implemented with multiple locks similar as in java.util.concurrent.ConcurrentHashMap, where each lock guards one part of the table. The maps used on clients do not need concurrency and are not run with locks to avoid unnecessary overhead.

D. Internode Communication

The system is divided into multiple clients and one server to support anonymization of datasets scattered or generated on different nodes across the network or cluster. The server node contains shared data needed to perform unique masking across multiple nodes. To comply with the in-situ idea, when masking a value, only if the specific masking technique does not find necessary data locally, it requests it from the server node. For example, when pseudonymizing string value by using the substitution technique, if the client node does not find the mapping in its local map, the server node is queried for that mapping. The server node looks up the key in its map and returns it if it is found, otherwise the mapping is generated. The client node stores the received mapping in its local map to avoid the future query to the server. In case no maps specified by the configuration are found locally when the node is run, the client requests the maps from the server. Map files are quickly transferred as data streams.

Finagle [15] asynchronous Remote Procedure Call (RPC) system is used for client server communication. Finagle server exposes callable RPC services com.twitter.finagle.Service for each masker. The client handles asynchronous requests to the server by com.twitter.util.Future objects, which invoke registered callbacks when the result from the server arrives. Thus, the masking blocks for rows of data, not for each field.

E. Integration into Apache Storm

We integrated anonymization system into Storm [16] - a distributed, fault-tolerant system for processing streams of data. Its topology comprises two main types of nodes – spouts and bolts, which are scattered across the network cluster and perform data reading/writing and data processing, respectively.

As depicted in Figure 2, input into the topology is done by spout node type, which is subscribed to Kafka [17] Topic to get lines of text to anonymize. Kafka is a messaging queue distributed on a cluster, accompanying Storm with a messaging system. The input is then parsed in LineSplitterBolt node and sent down the topology into multiple ClientDataMaskingBolt nodes in form of tuples. Tuples contain sets of values to anonymize. Each such node functions as a client in the masking process and contacts the ServerDataMaskingBolt if the masking technique requires so. ServerDataMaskingBolt node

Figure 2  Masking clients inside multiple Storm bolts
provides access to one or more maps, same as in the non-Storm masking cluster. Each ClientDataMaskingBolt is provided with the details of masking server location before the topology is submitted. The Storm topology is defined in a java program and automatically submitted to the cluster from any cluster node. Anonymized data is passed from each running ClientDataMaskingBolt bolt to KafkaSinkBolt, which writes the output.

F. Configuration

Configuration for client part of the process comprises description of input (file format, database table connection, messaging queue parameters) and specification of dataset columns to anonymize (column index, type of anonymization, masker id and masker seed). For the server part of the process it contains description of the server-side maskers (masker id, masker seed and masker specific parameters). When running the system on a single node, configurations are passed as XML files from the command line. In case of both in-situ processing or processing on Storm, client and server anonymization nodes obtain configuration from Zookeeper [18] - a centralized service for maintaining configuration information and distributed synchronization. The configuration from the XML file is stored into Zookeeper by passing path to XML configuration files to Zookeeper’s store similar to file system by ‘zkcli.sh –cmd upconfig –zhhost 127.0.0.1:2181 –d /path/to/config/files –a anonymization’ script. Uploaded configuration files are retrievable from the running Zookeeper service like ‘/configs/anonymization/client_config.xml’.

G. Dataset and Measurements

We used real telco Call Data Record (CDR) dataset for testing the performance of the anonymization system. CDR data is the fundamental subject of analysis such as churn prevention and user segmentation in telecoms. It is generated by telecoms’ equipment to record the details of the phone call that is made through that specific device. As it is generated in different places of a distributed environment, it is suitable for in-situ data processing. Also, analysis of customer data raises concerns about privacy and demands anonymization.

The sample CDR dataset used for testing comprises approximately 465 GB of post-paid and 73 GB of pre-paid customer data records for a period of 6 months. The dataset is stored in multiple CSV text files and comprises attributes like id, call start and end time, geographic location, duration, cost, event type, tariff, etc. For post-paid data, we chose to mask fields BILLED_MSISDN, CALLING_NUMBER, CALLED_NUMBER and OTHER_PARTY_TEL_NUMBER with partial substitution masking technique using the same server-side masker. Similarly, we masked fields MSISDN_NUMBER and B_NUMBER of the pre-paid dataset with partial substitution technique. Thus, the same partial substitution masker was used to mask the chosen fields and provide consistency across two independent dataset.

Figure 3 Frequency of masking requests from client to server

The effect of in-situ processing is depicted in Figure 3, where it is shown that clients' requests to server exponentially decrease as data fields are anonymized. In Phase 1, phone numbers (in reality generated at different telecom base stations) are unknown and requests to the server are frequent. Later in Phase 2, no new phone numbers appear and there is no need for requests to the server because the client masking nodes (in reality located on base stations) contain all the mapping data.

In everyday use of distributed anonymization system, impact of Phase 1 communication can be minimized by using pre built memory maps, or utilizing historical dataset on server side with only gathering distinct values to speed-up memory map buildup. After that, in-situ processing on all distributed nodes can be started and behaviour of whole system from that point will be only aligned to Phase 2 of internode communication.

After anonymizing more than 3 billion CDR records, the map contained 21,308,495 distinct phone number mappings. Original dataset didn’t have standardized phone numbers, thus same number could appear multiple times within masked value set depending on formatting of that specific number (e.g. with country prefix, with city prefix, shortcode, etc.). Standardization of phone numbers was out of scope for this paper.

Actual test results compared towards existing (and non-distributed) commercial Enterprise Data Masking solutions were omitted from this paper because we didn't get necessary approvals in due time.
IV. CONCLUSION

Anonymization is becoming more important with growing volume of generated data and privacy protection concerns. Newer anonymization techniques are required to resist the background knowledge and linking attacks that are possible by versatile datasets available online. Also, the techniques need to be built into processing systems capable of handling high volume and high velocity of Big Data. Our work presents a system suitable for in-situ anonymization of data in places where it was generated. It is appropriate for high volume of data with masking techniques’ appropriate mapping data structures. Moreover, latencies and unexpected pauses are minimized to adhere to the requirements of high velocity real time data. Also, internode communication in a distributed environment is minimized. Beside from the in-situ processing architecture, the system is also used in Storm distributed realtime computation system. In further work, datasets from other domains will be anonymized and distributed environments of different sizes and data generation speeds will be tested.