A Scalable Two-Phase Top-Down Specialization Approach for Data Anonymization Using MapReduce on Cloud

Xuyun Zhang, Laurence T. Yang, *Senior Member*, *IEEE*, Chang Liu, and Jinjun Chen, *Member*, *IEEE*

Abstract—A large number of cloud services require users to share private data like electronic health records for data analysis or mining, bringing privacy concerns. Anonymizing data sets via generalization to satisfy certain privacy requirements such as k-anonymity is a widely used category of privacy preserving techniques. At present, the scale of data in many cloud applications increases tremendously in accordance with the Big Data trend, thereby making it a challenge for commonly used software tools to capture, manage, and process such large-scale data within a tolerable elapsed time. As a result, it is a challenge for existing anonymization approaches to achieve privacy preservation on privacy-sensitive large-scale data sets due to their insufficiency of scalability. In this paper, we propose a scalable two-phase top-down specialization (TDS) approach to anonymize large-scale data sets using the MapReduce framework on cloud. In both phases of our approach, we deliberately design a group of innovative MapReduce jobs to concretely accomplish the specialization computation in a highly scalable way. Experimental evaluation results demonstrate that with our approach, the scalability and efficiency of TDS can be significantly improved over existing approaches.

Index Terms—Data anonymization, top-down specialization, MapReduce, cloud, privacy preservation

1 Introduction

Cloud computing, a disruptive trend at present, poses a significant impact on current IT industry and research communities [1], [2], [3]. Cloud computing provides massive computation power and storage capacity via utilizing a large number of commodity computers together, enabling users to deploy applications cost-effectively without heavy infrastructure investment. Cloud users can reduce huge upfront investment of IT infrastructure, and concentrate on their own core business. However, numerous potential customers are still hesitant to take advantage of cloud due to privacy and security concerns [4], [5]. The research on cloud privacy and security has come to the picture [6], [7], [8], [9].

Privacy is one of the most concerned issues in cloud computing, and the concern aggravates in the context of cloud computing although some privacy issues are not new [1], [5]. Personal data like electronic health records and financial transaction records are usually deemed extremely sensitive although these data can offer significant human benefits if they are analyzed and mined by organizations such as disease research centres. For instance, Microsoft HealthVault [10], an online cloud health service, aggregates data from users and shares the data with research institutes. Data privacy can be divulged with less effort by malicious cloud users or providers because of the failures of some traditional privacy protection measures on cloud [5]. This can bring considerable economic loss or severe social reputation impairment to data owners. Hence, data privacy issues need to be addressed urgently before data sets are analyzed or shared on cloud.

Data anonymization has been extensively studied and widely adopted for data privacy preservation in noninteractive data publishing and sharing scenarios [11]. Data anonymization refers to hiding identity and/or sensitive data for owners of data records. Then, the privacy of an individual can be effectively preserved while certain aggregate information is exposed to data users for diverse analysis and mining. A variety of anonymization algorithms with different anonymization operations have been proposed [12], [13], [14], [15]. However, the scale of data sets that need anonymizing in some cloud applications increases tremendously in accordance with the cloud computing and Big Data trends [1], [16]. Data sets have become so large that anonymizing such data sets is becoming a considerable challenge for traditional anonymization algorithms. The researchers have begun to investigate the scalability problem of large-scale data anonymization [17], [18].

Large-scale data processing frameworks like MapReduce [19] have been integrated with cloud to provide powerful computation capability for applications. So, it is promising

- *X. Zhang is with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China, and the Faculty of Engineering and IT, University of Technology, PO Box 123, Broadway, Sydney, NSW 2007, Australia. E-mail: xyzhanggz@gmail.com.*
- *L.T. Yang is with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China, and the Department of Computer Science, St. Francis Xavier University, Annex 11B, Antigonish, NS B2G 2W5, Canada. E-mail: ltyang@stfx.ca.*
- *C. Liu and J. Chen are with the Faculty of Engineering and IT, University of Technology, PO Box 123, Broadway, Sydney, NSW 2007, Australia. E-mail: {changliu.it, jinjun.chen}@gmail.com.*

Manuscript received 16 Sept. 2012; revised 19 Jan. 2013; accepted 6 Feb. 2013; published online 22 Feb. 2013.
Recommended for acceptance by X. Li, P. McDaniel, R. Poovendran, and G. Wang.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDSSI-2012-09-0909.
Digital Object Identifier no. 10.1109/TPDS.2013.48.

to adopt such frameworks to address the scalability problem of anonymizing large-scale data for privacy preservation. In our research, we leverage MapReduce, a widely adopted parallel data processing framework, to address the scalability problem of the top-down specialization (TDS) approach [12] for large-scale data anonymization. The TDS approach, offering a good tradeoff between data utility and data consistency, is widely applied for data anonymization [12], [20], [21], [22]. Most TDS algorithms are centralized, resulting in their inadequacy in handling large-scale data sets. Although some distributed algorithms have been proposed [20], [22], they mainly focus on secure anonymization of data sets from multiple parties, rather than the scalability aspect. As the MapReduce computation paradigm is relatively simple, it is still a challenge to design proper MapReduce jobs for TDS.

In this paper, we propose a highly scalable two-phase TDS approach for data anonymization based on MapReduce on cloud. To make full use of the parallel capability of MapReduce on cloud, specializations required in an anonymization process are split into two phases. In the first one, original data sets are partitioned into a group of smaller data sets, and these data sets are anonymized in parallel, producing intermediate results. In the second one, the intermediate results are integrated into one, and further anonymized to achieve consistent k-anonymous [23] data sets. We leverage MapReduce to accomplish the concrete computation in both phases. A group of MapReduce jobs is deliberately designed and coordinated to perform specializations on data sets collaboratively. We evaluate our approach by conducting experiments on real-world data sets. Experimental results demonstrate that with our approach, the scalability and efficiency of TDS can be improved significantly over existing approaches.

The major contributions of our research are threefold. First, we creatively apply MapReduce on cloud to TDS for data anonymization and deliberately design a group of innovative MapReduce jobs to concretely accomplish the specializations in a highly scalable fashion. Second, we propose a two-phase TDS approach to gain high scalability via allowing specializations to be conducted on multiple data partitions in parallel during the first phase. Third, experimental results show that our approach can significantly improve the scalability and efficiency of TDS for data anonymization over existing approaches.

The remainder of this paper is organized as follows: The next section reviews related work, and analyzes the scalability problem in existing TDS algorithms. In Section 3, we briefly present preliminary for our approach. Section 4 formulates the two-phase TDS approach, and Section 5 elaborates algorithmic details of MapReduce jobs. We empirically evaluate our approach in Section 6. Finally, we conclude this paper and discuss future work in Section 7.

2 Related Work and Problem Analysis

2.1 Related Work

Recently, data privacy preservation has been extensively investigated [11]. We briefly review related work below.

LeFevre et al. [17] addressed the scalability problem of anonymization algorithms via introducing scalable decision trees and sampling techniques. Iwuchukwu and Naughton [18] proposed an R-tree index-based approach by building a spatial index over data sets, achieving high efficiency. However, the above approaches aim at multidimensional generalization [15], thereby failing to work in the TDS approach. Fung et al. [12], [20], [21] proposed the TDS approach that produces anonymous data sets without the data exploration problem [11]. A data structure Taxonomy Indexed PartitionS (TIPS) is exploited to improve the efficiency of TDS. But the approach is centralized, leading to its inadequacy in handling large-scale data sets.

Several distributed algorithms are proposed to preserve privacy of multiple data sets retained by multiple parties. Jiang and Clifton [24] and Mohammed et al. [22] proposed distributed algorithms to anonymize vertically partitioned data from different data sources without disclosing privacy information from one party to another. Jurczyk and Xiong [25] and Mohammed et al. [20] proposed distributed algorithms to anonymize horizontally partitioned data sets retained by multiple holders. However, the above distributed algorithms mainly aim at securely integrating and anonymizing multiple data sources. Our research mainly focuses on the scalability issue of TDS anonymization, and is, therefore, orthogonal and complementary to them.

As to MapReduce-relevant privacy protection, Roy et al. [26] investigated the data privacy problem caused by MapReduce and presented a system named *Airavat* incorporating mandatory access control with differential privacy. Further, Zhang et al. [27] leveraged MapReduce to automatically partition a computing job in terms of data security levels, protecting data privacy in hybrid cloud. Our research exploits MapRedue itself to anonymize large-scale data sets before data are further processed by other MapReduce jobs, arriving at privacy preservation.

2.2 Problem Analysis

We analyze the scalability problem of existing TDS approaches when handling large-scale data sets on cloud.

The centralized TDS approaches in [12], [20], and [21] exploits the data structure TIPS to improve the scalability and efficiency by indexing anonymous data records and retaining statistical information in TIPS. The data structure speeds up the specialization process because indexing structure avoids frequently scanning entire data sets and storing statistical results circumvents recomputation overheads. On the other hand, the amount of metadata retained to maintain the statistical information and linkage information of record partitions is relatively large compared with data sets themselves, thereby consuming considerable memory. Moreover, the overheads incurred by maintaining the linkage structure and updating the statistic information will be huge when date sets become large. Hence, centralized approaches probably suffer from low efficiency and scalability when handling large-scale data sets.

There is an assumption that all data processed should fit in memory for the centralized approaches [12]. Unfortunately, this assumption often fails to hold in most data-intensive cloud applications nowadays. In cloud environments, computation is provisioned in the form of

virtual machines (VMs). Usually, cloud compute services offer several flavors of VMs. As a result, the centralized approaches are difficult in handling large-scale data sets well on cloud using just one single VM even if the VM has the highest computation and storage capability.

A distributed TDS approach [20] is proposed to address the distributed anonymization problem which mainly concerns privacy protection against other parties, rather than scalability issues. Further, the approach only employs information gain, rather than its combination with privacy loss, as the search metric when determining the best specializations. As pointed out in [12], a TDS algorithm without considering privacy loss probably chooses a specialization that leads to a quick violation of anonymity requirements. Hence, the distributed algorithm fails to produce anonymous data sets exposing the same data utility as centralized ones. Besides, the issues like communication protocols and fault tolerance must be kept in mind when designing such distributed algorithms. As such, it is inappropriate to leverage existing distributed algorithms to solve the scalability problem of TDS.

3 Preliminary

3.1 Basic Notations

We describe several basic notations for convenience. Let D denote a data set containing data records. A record $r \in D$ has the form $r = \langle v_1, v_2, \ldots, v_m, sv \rangle$, where m is the number of attributes, v_i, $1 \leq i \leq m$, is an attribute value and sv is a sensitive value like diagnosis. The set of sensitive values is denoted as SV. An attribute of a record is denoted as $Attr$, and the taxonomy tree of this attribute is denoted as TT. Let DOM represent the set of all domain values in TT. The quasi-identifier of a record is denoted as $qid = \langle q_1, q_2, \ldots, q_m \rangle$, where $q_i \in DOM_i$. Quasi-identifiers, representing groups of anonymous records, can lead to privacy breach if they are too specific that only a small group of people are linked to them [11]. Quasi-identifier set is denoted as $QID = \langle Attr_1, Attr_2, \ldots, Attr_m \rangle$. The set of the records with qid is defined as QI-group [28], denoted by $QIG(qid)$. QI is the acronym of quasi-identifier.

Without loss of generality, we adopt k-anonymity [23] as the privacy model herein, i.e., for any $qid \in QID$, the size of $G(qid)$ must be zero or at least k. Otherwise, the individuals owning such a quasi-identifier can be linked to sensitive information with higher confidence than expected, resulting in privacy breach. The k-anonymity privacy model can combat such a privacy breach because it ensures that an individual will not be distinguished from other at least $k-1$ ones. The anonymity parameter k is specified by users according to their privacy requirements.

In the TDS approach, a data set is anonymized via performing specialization operations. A specialization operation is to replace a domain value with all its child values. Formally, a specialization operation is represented as $spec : p \rightarrow Child(p)$, where p is a domain value and $Child(p) \subseteq DOM$ is the set of all the child values of p. The domain values of an attribute form a "cut" through its taxonomy tree [11]. The cut of attribute $Attr_i$, denoted as Cut_i, $1 \leq i \leq m$, is a subset of values in $DOM_i \cdot Cut_i$ contains exactly one value in each root-to-leaf path in taxonomy tree TT_i. The cuts of all attributes determine the anonymity of a data set. To capture the degree of anonymization intuitively during the specialization process, we give the subsequent definition.

Definition 1. (Anonymization Level). *A vector of cuts of all attributes is defined as anonymization level, denoted as AL. Formally, $AL = \langle Cut_1, Cut_2, \ldots, Cut_m \rangle$, where Cut_i, $1 \leq i \leq m$ is the cut of taxonomy tree TT_i.*

Anonymization level can intuitively represent the anonymization degree of a data set, i.e., the more specific AL a data set has, the less degree of anonymization it corresponds to. Thus, TDS approaches employ anonymization level to track and manage the specialization process.

MapReduce notations can be found in Appendix A.1, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2013.48, (All appendices are included in the supplemental file).

3.2 Top-Down Specialization

Generally, TDS is an iterative process starting from the topmost domain values in the taxonomy trees of attributes. Each round of iteration consists of three main steps, namely, finding the best specialization, performing specialization and updating values of the search metric for the next round [12]. Such a process is repeated until k-anonymity is violated, to expose the maximum data utility. The goodness of a specialization is measured by a search metric. We adopt the information gain per privacy loss (IGPL), a tradeoff metric that considers both the privacy and information requirements, as the search metric in our approach. A specialization with the highest IGPL value is regarded as the best one and selected in each round. We briefly describe how to calculate the value of IGPL subsequently to make readers understand our approach well. Interested readers can refer to [11] for more details.

Given a specialization $spec : p \rightarrow Child(p)$, the IGPL of the specialization is calculated by

$$IGPL(spec) = IG(spec)/(PL(spec) + 1). \quad (1)$$

The term $IG(spec)$ is the information gain after performing $spec$, and $PL(spec)$ is the privacy loss. $IG(spec)$ and $PL(spec)$ can be computed via statistical information derived from data sets. Let R_x denote the set of original records containing attribute values that can be generalized to x. $|R_x|$ is the number of data records in R_x. Let $I(R_x)$ be the entropy of R_x. Then, $IG(spec)$ is calculated by

$$IG(spec) = I(R_p) - \sum_{c \in Child(p)} \left(\frac{|R_c|}{|R_p|} \right) I(R_c), \quad (2)$$

Let $|(R_x, sv)|$ denote the number of the data records with sensitive value sv in R_x. $I(R_x)$ is computed by

$$I(R_x) = - \sum_{sv \in SV} \left(\frac{|(R_x, sv)|}{|R_x|} \right) . \log_2 \left(\frac{|R_x, sv|}{|R_x|} \right). \quad (3)$$

The anonymity of a data set is defined by the minimum group size out of all QI-groups, denoted as A, i.e., $A = \min_{qid \in QID} \{|QIG(qid)|\}$, where $|QIG(qid)|$ is the size of

$QIG(qid)$. Let $A_p(spec)$ denote the anonymity before performing $spec$, while $A_c(spec)$ be that after performing $spec$. Privacy loss caused by $spec$ is calculated by

$$PL(spec) = A_p(spec) - A_c(spec). \quad (4)$$

4 Two-Phase Top-Down Specialization (TPTDS)

The sketch of the TPTDS approach is elaborated in Section 4.1. Three components of the TPTDS approach, namely, data partition, anonymization level merging, and data specialization are detailed in Sections 4.2, 4.3, and 4.4, respectively.

4.1 Sketch of Two-Phase Top-Down Specialization

We propose a TPTDS approach to conduct the computation required in TDS in a highly scalable and efficient fashion. The two phases of our approach are based on the two levels of parallelization provisioned by MapReduce on cloud. Basically, MapReduce on cloud has two levels of parallelization, i.e., job level and task level. Job level parallelization means that multiple MapReduce jobs can be executed simultaneously to make full use of cloud infrastructure resources. Combined with cloud, MapReduce becomes more powerful and elastic as cloud can offer infrastructure resources on demand, for example, Amazon Elastic MapReduce service [29]. Task level parallelization refers to that multiple mapper/reducer tasks in a MapReduce job are executed simultaneously over data splits. To achieve high scalability, we parallelizing multiple jobs on data partitions in the first phase, but the resultant anonymization levels are not identical. To obtain finally consistent anonymous data sets, the second phase is necessary to integrate the intermediate results and further anonymize entire data sets. Details are formulated as follows.

In the first phase, an original data set D is partitioned into smaller ones. Let D_i, $1 \le i \le p$, denote the data sets partitioned from D the, where p is the number of partitions, and $D = \sum_{i=1}^{p} D_i$, $D_i \cap D_j = \emptyset$, $1 \le i < j \le p$. The details of how to partition D will be discussed in Section 4.2.

Then, we run a subroutine over each of the partitioned data sets in parallel to make full use of the job level parallelization of MapReduce. The subroutine is a MapReduce version of centralized TDS (MRTDS) which concretely conducts the computation required in TPTDS. MRTDS anonymizes data partitions to generate intermediate anonymization levels. An intermediate anonymization level means that further specialization can be performed without violating k-anonymity. MRTDS only leverages the task level parallelization of MapReduce. More details of MRTDS will be elaborated in Section 5. Formally, let function $MRTDS(D, k, AL) \rightarrow AL'$ represent a MRTDS routine that anonymizes data set D to satisfy k-anonymity from anonymization level AL to AL'. Thus, a series of functions $MRTDS(D_i, k^I, AL^0) \rightarrow AL'_i$, $1 \le i \le p$, are executed simultaneously in the first phase. The term k^I denotes the intermediate anonymity parameter, usually given by application domain experts. Note that k^I should satisfy $k^I \ge k$ to ensure privacy preservation. AL^0 is the initial anonymization level, i.e., $AL^0 = \langle \{Top_1\}, \{Top_2\}, \ldots, \{Top_m\} \rangle$, where $Top_j \in DOM_j$, $1 \le j \le m$, is the topmost domain value in TT_i. AL'_i is the resultant intermediate anonymization level.

In the second phase, all intermediate anonymization levels are merged into one. The merged anonymization level is denoted as AL^I. The merging process is formally represented as function $merge(\langle AL'_1, AL'_2, \ldots, AL'_p \rangle) \rightarrow AL^I$. The function will be detailed in Section 4.3. Then, the whole data set D is further anonymized based on AL^I, achieving k-anonymity finally, i.e., $MRTDS(D, k, AL^I) \rightarrow AL^*$, where AL^* denotes the final anonymization level. Ultimately, D is concretely anonymized according to AL^*, described in Section 4.4. Above all, Algorithm 1 depicts the sketch of the two-phase TDS approach.

Algorithm 1. Sketch of Two-Phase TDS (TPTDS).

Input: Data set D, anonymity parameters k, k^I and the number of partitions p.

Output: Anonymous data set D^*.

1: Partition D into D_i,$1 \le i \le p$.
2: Execute $MRTDS(D_i, k^I, AL^0) \rightarrow AL'_i$, $1 \le i \le p$ in parallel as multiple MapReduce jobs.
3: Merge all intermediate anonymization levels into one, $merge(AL'_1, AL'_2, \ldots, AL'_p) \rightarrow AL^I$.
4: Execute $MRTDS(D, k, AL^I) \rightarrow AL^*$ to achieve k-anonymity.
5: Specialize D according to AL^*, Output D^*.

In essential, TPTDS divides specialization operations required for anonymization into the two phases. Let SP_{1i}, $1 \le i \le p$, denote the specialization sequence on D_i in the first phase, i.e., $SP_{1i} = \langle spec_{i1}, spec_{i2}, \ldots, spec_{ij_i} \rangle$, where j_i is the number of specializations. The first common subsequence of SP_{1i}, $1 \le i \le p$, is denoted as SP^I. Let SP_2 denote the specialization sequence in the second phase. SP_2 is determined by AL^I rather than k^I. Specifically, more specific AL^I implies smaller SP_2. Throughout TPTDS, the specializations in the set $SP^I \cup SP_2$ come into effect for anonymization. The specializations in the set $SP^{Extra} = (\bigcup_{i=1}^{p} SP_{1i})/SP^I$ are extra overheads introduced by TPTDS.

The influence of p and k^I on the efficiency is analyzed as follows. Greater p means higher degree of parallelization in the first phase, and less k^I indicates more computation is conducted in the first phase. Thus, greater p and less k^I can improve the efficiency. However, greater p and less k^I probably lead to larger SP^{Extra}, thereby degrading the overall efficiency. Usually, greater p causes smaller SP^I and larger $\bigcup_{i=1}^{p} SP_{1i}$, and less k^I result in larger SP_{1i}.

The basic idea of TPTDS is to gain high scalability by making a tradeoff between scalability and data utility. We expect that slight decrease of data utility can lead to high scalability. The influence of p and k^I on the data utility is analyzed as follows. The data utility produced via TPTDS is roughly determined by $SP^I \cup SP_2$. Greater p means that the specializations in SP^I are selected according to IGPL values from smaller data sets, resulting in exposing less data utility. However, greater p also implies smaller SP^I but larger SP_2, which means more data utility can be produced because specializations in SP_2 are selected according an entire data set. Larger k^I indicates larger SP_2, generating more data utility.

In terms of the above analysis, the optimization of the trade-off between scalability and data utility can be fulfilled

by tuning p and k^I. It is hard to quantitatively formulate the relationships between TPTDS performance and the two parameters due to they are data set content specific. But users can leverage the qualitative relationships analyzed above to tune performance heuristically.

4.2 Data Partition

When D is partitioned into $D_i, 1 \leq i \leq p$, it is required that the distribution of data records in D_i is similar to D. A data record here can be treated as a point in an m-dimension space, where m is the number of attributes. Thus, the intermediate anonymization levels derived from $D_i, 1 \leq i \leq p$, can be more similar so that we can get a better merged anonymization level. Random sampling technique is adopted to partition D, which can satisfy the above requirement. Specifically, a random number $rand$, $1 \leq rand \leq p$, is generated for each data record. A record is assigned to the partition D_{rand}. Algorithm 2 shows the MapReduce program of data partition. Note that the number of *Reducers* should be equal to p, so that each *Reducer* handles one value of $rand$, exactly producing p resultant files. Each file contains a random sample of D.

ALGORITHM 2. DATA PARTITION MAP & REDUCE.
Input: Data record (ID_r, r), $r \in D$, partition parameter p.
Output: D_i, $1 \leq i \leq p$.
Map: Generate a random number $rand$, where $1 \leq rand \leq p$; emit ($rand$, r).
Reduce: For each rand, emit (*null*, *list*(r)).

Once partitioned data sets $D_i, 1 \leq i \leq p$, are obtained, we run $MRTDS(D_i, k_I, AL^0)$ on these data sets in parallel to derive intermediate anonymization levels $AL_i^*, 1 \leq i \leq p$.

4.3 Anonymization Level Merging

All intermediate anonymization levels are merged into one in the second phase. The merging of anonymization levels is completed by merging cuts. Specifically, let Cut_a in AL'_a and Cut_b in AL'_b be two cuts of an attribute. There exist domain values $q_a \in Cut_a$ and $q_b \in Cut_b$ that satisfy one of the three conditions: q_a is identical to q_b, q_a is more general than q_b, or q_a is more specific than q_b. To ensure that the merged intermediate anonymization level AL^I never violates privacy requirements, the more general one is selected as the merged one, for example, q_a will be selected if q_a is more general than or identical to q_b. For the case of multiple anonymization levels, we can merge them in the same way iteratively. The following lemma ensures that AL^I still complies privacy requirements.

Lemma 1. *If intermediate anonymization levels $AL'_i, 1 \leq i \leq p$, satisfy k^I-anonymity, the merged intermediate anonymization level AL^I will satisfies k'-anonymity, where $AL^I \leftarrow merge(\langle AL'_1, AL'_2, \ldots, AL'_p \rangle)$, $k' \geq k^I$.*

The proof of Lemma 1 can be found in Appendix B.1, which is available in the online supplemental material. Our approach can ensure the degree of data privacy preservation, as TPTDS produces k-anonymous data sets finally. Lemma 1 ensures that the first phase produces consistent anonymous data sets that satisfy higher degree of privacy preservation than users' specification. Then, MRTDS can further anonymize the entire data sets to produce final k-anonymous data sets in the second phase.

4.4 Data Specialization

An original data set D is concretely specialized for anonymization in a one-pass MapReduce job. After obtaining the merged intermediate anonymization level AL^I, we run $MRTDS(D, k, AL^I)$ on the entire data set D, and get the final anonymization level AL^*. Then, the data set D is anonymized by replacing original attribute values in D with the responding domain values in AL^*.

Details of *Map* and *Reduce* functions of the data specialization MapReduce job are described in Algorithm 3. The *Map* function emits anonymous records and its count. The *Reduce* function simply aggregates these anonymous records and counts their number. An anonymous record and its count represent a QI-group. The QI-groups constitute the final anonymous data sets.

ALGORITHM 3. DATA SPECIALIZATION MAP & REDUCE.
Input: Data record (ID_r, r), $r \in D$. ; Anonymization level AL^*.
Output: Anonymous record (r^*, $count$).
Map: Construct anonymous record $r^* = p_1, \langle p_2, \ldots, p_m, sv \rangle$, p_i, $1 \leq i \leq m$, is the parent of a specialization in current AL and is also an ancestor of v_i in r; emit ($r^*, count$).
Reduce: For each r^*, $sum \leftarrow \sum count$; emit ($r^*$, sum).

5 MAPREDUCE VERSION OF CENTRALIZED TDS

We elaborate the MRTDS in this section. MRTDS plays a core role in the two-phase TDS approach, as it is invoked in both phases to concretely conduct computation. Basically, a practical MapReduce program consists of *Map* and *Reduce* functions, and a *Driver* that coordinates the macro execution of jobs. In Section 5.1, we describe the MRTDS *Driver*. The *Map* and *Reduce* functions are detailed in Sections 5.2 and 5.3. Finally, we present the implementation in Section 5.4.

5.1 MRTDS Driver

Usually, a single MapReduce job is inadequate to accomplish a complex task in many applications. Thus, a group of MapReduce jobs are orchestrated in a driver program to achieve such an objective. MRTDS consists of *MRTDS Driver* and two types of jobs, i.e., *IGPL Initialization* and *IGPL Update*. The driver arranges the execution of jobs.

Algorithm 4 frames *MRTDS Driver* where a data set is anonymized by TDS. It is the algorithmic design of function $MRTDS(D, k, AL) \rightarrow AL'$ mentioned in Section 4.2. Note that we leverage anonymization level to manage the process of anonymization. Step 1 initializes the values of information gain and privacy loss for all specializations, which can be done by the job *IGPL Initialization*.

ALGORITHM 4. MRTDS DRIVER.
Input: Data set D, anonymization level AL and k-anonymity parameter k.
Output: Anonymization level AL'.
1: Initialize the values of search metric IGPL, i.e., for each specialization $spec \in \cup_{j=1}^{m} Cut_j$. The IGPL value of $spec$ is computed by job *IGPL Initialization*.

2: **while** $\exists\ spec \in \cup_{j=1}^{m} Cut_j$ is valid
  2.1: Find the best specialization from AL_i, $spec_{Best}$.
  2.2: Update AL_i to AL_{i+1}.
  2.3: Update information gain of the new specializations in AL_{i+1}, and privacy loss for each specialization via job *IGPL Update*.
  end while
$AL' \leftarrow AL$.

Step 2 is iterative. First, the best specialization is selected from valid specializations in current anonymization level as described in Step 2.1. A specialization $spec$ is a valid one if it satisfies two conditions. One is that its parent value is not a leaf, and the other is that the anonymity $A_c(spec) > k$, i.e., the data set is still k-anonymous if $spec$ is performed. Then, the current anonymization level is modified via performing the best specialization in Step 2.2, i.e., removing the old specialization and inserting new ones that are derived from the old one. In Step 2.3, information gain of the newly added specializations and privacy loss of all specializations need to be recomputed, which are accomplished by job *IGPL Update*. The iteration continues until all specializations become invalid, achieving the maximum data utility.

MRTDS produces the same anonymous data as the centralized TDS in [12], because they follow the same steps. MTRDS mainly differs from centralized TDS on calculating IGPL values. However, calculating IGPL values dominates the scalability of TDS approaches, as it requires TDS algorithms to count the statistical information of data sets iteratively. MRTDS exploits MapReduce on cloud to make the computation of IGPL parallel and scalable. We present *IGPL Initialization* and *IGPL Update* subsequently.

5.2 IGPL Initialization Job

The *Map* and *Reduce* functions of the job *IGPL Initialization* are described in Algorithms 5 and 6, respectively. The main task of *IGPL Initialization* is to initialize information gain and privacy loss of all specializations in the initial anonymization level AL. According to (2) and (3), the statistical information $|R_p|$, $|(R_p, sv)|$, $|R_c|$, and $|(R_c, sv)|$ is required for each specialization to calculate information gain. In terms of (4), the number of records in each current QI-group needs computing, so does the number of records in each QI-group after potential specializations.

ALGORITHM 5. IGPL INITIALIZATION MAP.
Input: Data record (ID_r, r), $r \in D$; anonymization level AL.
Output: Intermediate key-value pair (key, $count$).
1: For each attribute value v_i in r, find its specialization in current AL: spec. Let p be the parent in $spec$, and c be the p's child value that is also an ancestor of v_i in TT_i.
2: For each v_i, emit $(\langle p, c, sv\rangle, count)$.
3: Construct quasi-identifier $qid^* = \langle p_1, p_2, \ldots, p_m\rangle$, where p_i, $1 \leq \mathrm{i} \leq m$, is the parent of a specialization in current AL. Emit$(\langle qid^*, \$, \#\rangle, count)$.
4: For each $i \in [1, m]$, replace p_i in qid^* with its child c_i, where c_i is also the ancestor of v_i. Let the resultant quasi-identifier be qid. Emit$(\langle qid, p_i, \#\rangle, count)$.

Algorithm 5 describes the *Map* function. The input is data sets that consist of a number of records. ID_r is the sequence number of the record r. Steps 1 and 2 are to compute $|R_p|$, $|(R_p, sv)|$, $|R_c|$, and $|(R_c, sv)|$. Step 1 gets the potential specialization for the attribute values in r. Then Step 2 emits key-value pairs containing the information of specialization, sensitive value, and the count information of this record. According to the above information, we compute information gain for a potential specialization in the corresponding *Reduce* function. Step 3 aims at computing the current anonymity $A_p(spec)$, while Step 4 is to compute anonymity $A_c(spec)$ after potential specializations. The symbol "#" is used to identify whether a key is emitted to compute information gain or anonymity loss, while the symbol "$" is employed to differentiate the cases whether a key is for computing $A_p(spec)$ or $A_c(spec)$.

Algorithm 6 specifies the *Reduce* function. The first step is to accumulate the values for each input key. If a key is for computing information gain, then the corresponding statistical information is updated in Step 2.1. $I(R_p)$, $I(R_c)$, and $IG(spec)$ are calculated if all the count information they need has been computed in Steps 2.2 and 2.3 in terms of (2) and (3). A salient MapReduce feature that intermediate key-value pairs are sorted in the shuffle phase makes the computation of $IG(spec)$ sequential with respect to the order of specializations arriving at the same reducer. Hence, the reducer just needs to keep statistical information for one specialization at a time, which makes the reduce algorithm highly scalable.

ALGORITHM 6. IGPL INITIALIZATION REDUCE.
Input: Intermediate key-value pair (key, $list(count)$).
Output: Information gain ($spec, IG(spec)$) and anonymity ($spec$, $A_c(spec)$), ($spec$, $A_p(spec)$) for all specializations.
1: For each key, $sum \leftarrow \sum count$.
2: For each *key*, if $key.sv \neq \#$, update statistical counts:
  2.1: $|(R_c, sv)| \leftarrow sum$, $|R_c| \leftarrow sum + |R_c|$, $|(R_p, sv)| \leftarrow sum + |(R_p, sv)|$, $|R_p| \leftarrow sum + |R_p|$.
  2.2: If all sensitive values for child c have arrived, compute $I(R_c)$ according to (3).
  2.3: If all children c for parent p have arrived, compute $I(R_p)$ and $IG(spec)$. Emit $(spec, IG(spec))$.
3: For each *key*, if $key.sv = \#$, update anonymity.
  3.1: If $key.c = \$$ and $sum < A_p(spec)$, update current anonymity: $A_p(spec) \leftarrow sum$.
  3.2: If $key.c \neq \$$ and $sum < A_c(spec)$, update potential anonymity of $spec$: $A_c(spec) \leftarrow sum$.
4: Emit ($spec$, $A_p(spec)$) and emit ($spec$, $A_c(spec)$).

To compute the anonymity of data sets before and after a specialization, Step 3.1 finds the smallest number of records out of all current QI-groups, and Step 3.2 finds all the smallest number of records out of all potential QI-groups for each specialization. Step 4 emits the results of anonymity. Note that there may be more than one key-value pair ($spec$, $A(spec)$) for one specialization in output files if more than one reducer is set. But we can find the smallest anonymity value in the driver program. Then in terms of (4), the privacy loss $PL(spec)$ is computed. Finally, $IGPL(spec)$ for each specialization is obtained by (1).

5.3 IGPL Update Job

The *IGPL Update* job dominates the scalability and efficiency of MRTDS, since it is executed iteratively as described in Algorithm 4. So far, iterative MapReduce jobs have not been

well supported by standard MapReduce framework like Hadoop [30]. Accordingly, Hadoop variations like Haloop [31] and Twister [32] have been proposed recently to support efficient iterative MapReduce computation. Our approach is based on the standard MapReduce framework to facilitate the discussion herein.

The *IGPL Update* job is quite similar to *IGPL Initialization*, except that it requires less computation and consumes less network bandwidth. Thus, the former is more efficient than the latter. Algorithm 7 describes the *Map* function of *IGPL Update*. The *Reduce* function is the same as *IGPL Initialization*, already described in Algorithm 3.

ALGORITHM 7. IGPL UPDATE MAP.
Input: Data record (ID_r, r), $r \in D$; Anonymization level AL.
Output: Intermediate key-value pair (key, $count$).
1: Let $attr$ be the attribute of the last best specialization. The value of this attribute in r is v. Find its specialization in current AL: spec. Let p be the parent in $spec$, and c be p's child that is also an ancestor of v; Emit $(\langle p, c, sv\rangle, count)$.
2: Construct quasi-identifier $qid^* = \langle p_1, p_2, \ldots, p_m\rangle$, $\mathrm{p_i}$, $1 \le i \le m$, is the parent of a specialization in current AL and is also an ancestor of v_i in r.
3: For each $i \in [1, m]$, replace p_i in qid^* with its child c_i if the specialization related to p_i is valid, where c_i is also the ancestor of v_i. Let the resultant quasi-identifier be qid. Emit $(\langle qid, \mathrm{p_i}, \#\rangle, count)$.

After a specialization $spec$ is selected as the best candidate, it is required to compute the information gain for the new specializations derived from $spec$. So, Step 1 in Algorithm 7 only emits the key-value pairs for the new specializations, rather than all in Algorithm 5. Note that it is unnecessary to recompute the information gain of other specializations because conducting the selected specialization never affects the information gain of others. Compared with *IGPL Initialization*, only a part of data is processed and less network bandwidth is consumed.

On the contrary, the anonymity values of other specializations will be influenced with high probability because splitting QI-groups according to $spec$ changes the minimality of the smallest QI-group in last round. Therefore, we need to compute $A_c(spec)$ for all specializations in AL, described in Step 2 and 3 of Algorithm 7. Yet $A_p(spec)$ can be directly obtained from the statistical information kept by the last best specialization. Note that if the specialization related to p_i in Step 3 is not valid, no resultant quasi-identifier will be created.

Since the *IGPL Update* job dominates the scalability and efficiency of MRTDS, we briefly analyze its complexity as follows. Let n denote all the records in a data set, m be the number of attributes, s be the number of mappers, and t be the number of reducers. As a mapper emits $(m + 1)$ key-value pairs, it takes $O(1)$ space and $O(m * n/s)$ time. Similarly, a reducer takes $O(1)$ space and $O(m * n/t)$ time. Note that a reducer only needs $O(1)$ space due to the MapReduce feature that the key-value pairs are sorted in the shuffle phase. Otherwise, the reducer needs more space to accumulate statistic information for a variety of specializations. The communication cost is $O(m * n)$ according to the map function, but communication traffics can be reduced heavily by optimization techniques like Combiner.

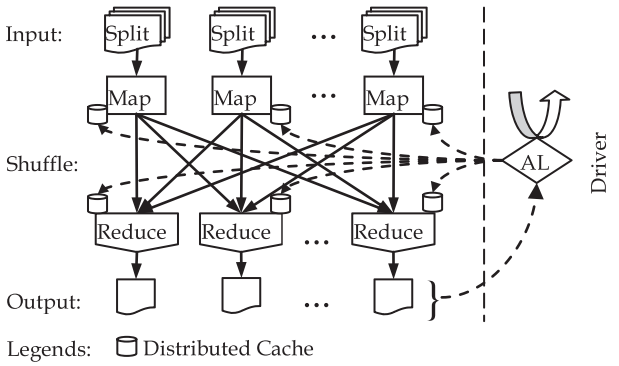


Fig. 1. Execution framework overview of MRTDS.

5.4 Implementation and Optimization

To elaborate how data sets are processed in MRTDS, the execution framework based on standard MapReduce is depicted in Fig. 1. The solid arrow lines represent the data flows in the canonical MapReduce framework. From Fig. 1, we can see that the iteration of MapReduce jobs is controlled by anonymization level AL in *Driver*. The data flows for handling iterations are denoted by dashed arrow lines. AL is dispatched from *Driver* to all workers including *Mappers* and *Reducers* via the distributed cache mechanism. The value of AL is modified in *Driver* according to the output of the *IGPL Initialization* or *IGPL Update* jobs. As the amount of such data is extremely small compared with data sets that will be anonymized, they can be efficiently transmitted between *Driver* and workers.

We adopt Hadoop [30], an open-source implementation of MapReduce, to implement MRTDS. Since most of *Map* and *Reduce* functions need to access current anonymization level AL, we use the distributed cache mechanism to pass the content of AL to each *Mapper* or *Reducer* node as shown in Fig. 1. Also, Hadoop provides the mechanism to set simple global variables for *Mappers* and *Reducers*. The best specialization is passed into the *Map* function of *IGPL Update* job in this way. The partition hash function in the shuffle phase is modified because the two jobs require that the key-value pairs with the same $key.p$ field rather than entire key should go to the same *Reducer*.

To reduce communication traffics, MRTDS exploits combiner mechanism that aggregates the key-value pairs with the same key into one on the nodes running *Map* functions. To further reduce the traffics, MD5 (Message Digest Algorithm) is employed to compress the records transmitted for anonymity computation, i.e., $(\langle MD5(qid), p_i, \#\rangle, count)$ is emitted in Step 4 of Algorithm 5 and Step 3 of Algorithm 7. $MD5(qid)$ is fixed-length and usually shorter than qid. As anonymity computation causes the most traffic as it emits m key-value pairs for each original record, this can considerably reduce network traffics. The *Reduce* function in Algorithm 6 can still correctly compute anonymity without being aware of the content of qid.

6 EVALUATION

6.1 Overall Comparison

To evaluate the effectiveness and efficiency of our two-phase approach, we compare it with the centralized TDS

approach proposed in [12], denoted as CentTDS. CentTDS is the state-of-the-art approach for TDS anonymization. Scalability and data utility are considered for the effectiveness. For scalability, we check whether both approaches can still work and scale over large-scale data sets. Data utility is measured by the metric *ILoss*, a general purpose data metric proposed in [28]. Literally, *ILoss* means information loss caused by data anonymization. Basically, higher *ILoss* indicates less data utility. How to calculate *ILoss* can be found in Appendix A.2, which is available in the online supplemental material. The *ILoss* of CentTDS and TPTDS are denoted as IL_{Cent} and IL_{TP}, respectively. The execution time of CentTDS and TPTDS are denoted as T_{Cent} and T_{TP}, respectively.

We roughly compare TPTDS and CentTDS as follows. TPTDS can scale over more computation nodes with the volume of data sets increasing, thereby gaining higher scalability. CentTDS will suffer from low scalability on large-scale data sets because it requires too much memory, while TPTDS can linearly scale over data sets of any size. Correspondingly, T_{TP} is often less than T_{Cent} for large-scale data sets. But note that, T_{Cent} can be less than T_{TP} due to extra overheads engendered by TPTDS when the scale of data sets or the MapReduce cluster is small. TPTDS is equivalent to MRTDS if parameter $p = 1$ or $k^I \geq k^{max}$, where k^{max} is the number of all records. As MRTDS produces the same anonymous data as centralized TDS, the value of IL_{TP} is equal to IL_{Cent} when $p = 1$ or $k^I \geq k^{max}$. In other cases, IL_{TP} is probably greater than IL_{Cent}, as some specializations selected in TPTDS are not globally optimal.

The overheads of our approach are mainly introduced by the MapReduce built-in operations and the parallelization in the first phase of TPTDS. Built-in MapReduce operations like data splitting and key-value pair sorting and transmission will cause overheads. The overheads are hard to quantitatively measure as they are implementation-, configuration-, and algorithm-specific. The extra specializations in the first phase incur overheads affecting the efficiency of TPTDS heavily. The relationship between the extra overheads and parameters p and k^I is qualitatively described in Section 4.1. The one-pass job partitioning original data sets also generates overheads.

6.2 Experiment Evaluation

6.2.1 Experiment Settings

Our experiments are conducted in a cloud environment named U-Cloud. U-Cloud is a cloud computing environment at the University of Technology Sydney (UTS). The system overview of U-Cloud has been depicted in Fig. 2. The computing facilities of this system are located among several labs at UTS. On top of hardware and Linux operating system (Ubuntu), we install KVM virtualization software [33] that virtualizes the infrastructure and provides unified computing and storage resources. To create virtualized data centers, we install OpenStack open source cloud environment [34] for global management, resource scheduling and interaction with users. Further, Hadoop [30] clusters are built based on the OpenStack cloud platform to facilitate large-scale data processing.

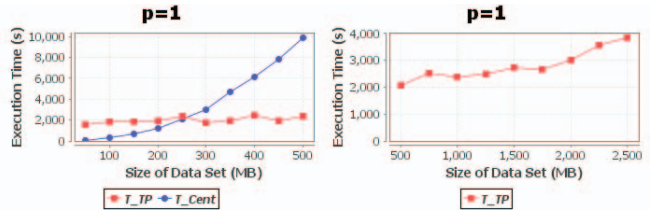


Fig. 2. Change of execution time w.r.t. data size: TPTDS versus CentTDS.

We use *Adult* data set [35], a public data set commonly used as a de facto benchmark for testing anonymization algorithms [12], [20]. We generate data sets by enlarging the *Adult* data set according to the approach in [20]. More details of experiment data are described in Appendix C.1, which is available in the online supplemental material.

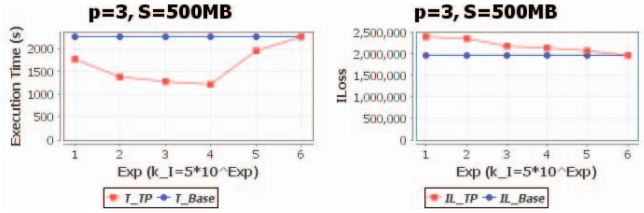
Both TPTDS and CentTDS are implemented in Java. Further, TPTDS is implemented with standard Hadoop MapReduce API and executed on a Hadoop cluster built on OpenStack. CentTDS is executed on a VM with type *m1.large*. The *m1.large* type has four virtual CPUs and 8-GB Memory. The maximum heap size of Java VM is set as 4 GB when running CentTDS. The Hadoop cluster consists of 20 VMs with type *m1.medium* which has two virtual CPUs and 4-GB Memory. The k-anonymity parameter is set as 50 throughout all experiments. Each round of experiment is repeated 20 times. The mean of the measured results is regarded as the representative.

6.2.2 Experiment Process and Results

We conduct three groups of experiments in this section to evaluate the effectiveness and efficiency of our approach. In the first one, we compare TPTDS with CentTDS from the perspectives of scalability and efficiency. In the other two, we investigate on the tradeoff between scalability and data utility via adjusting configurations. Generally, the execution time and *ILoss* are affected by three factors, namely, the size of a data set (S), the number of data partitions (p), and the intermediate anonymity parameter (k^I). How the three factors influence the execution time and *ILoss* of TPTDS is observed in the following experiments.

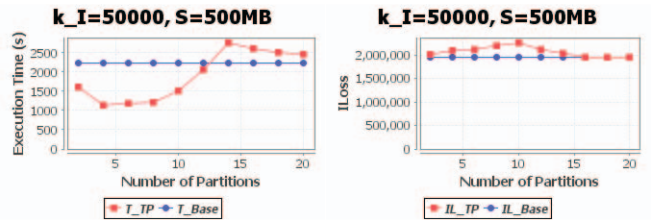
In the first group, we measure the change of execution time T_{Cent} and T_{TP} with respect to S when $p = 1$. The size S varies from 50 MB to 2.5 GB. The 2.5 GB data set contains nearly $2.5 * 10^7$ data records. The scale of data sets in our experiments is much greater than that in [12] and [20]. Thus, the data sets in our experiments are big enough to evaluate the effectiveness of our approach in terms of data volume or the number of data records. Note that $IL_{Cent} = IL_{TP}$ because TPTDS is equivalent to MRTDS when $p = 1$. So, we just demo the results of execution time. The results of the first group of experiments are depicted in Fig. 2.

Fig. 2a shows the change of T_{TP} and T_{Cent} with respect to the data size ranging from 50 to 500 MB. From Fig. 2a, we can see that both T_{TP} and T_{Cent} go up when data size increases although some slight fluctuations exist. The fluctuations are mainly caused by the content of data sets. T_{Cent} surges from tens of seconds to nearly 10,000 seconds,



(a) (b)

Fig. 3. Change of execution time and ILoss w.r.t. intermediate anonymity parameter.



(a) (b)

Fig. 4. Change in execution time and ILoss w.r.t. number of partitions.

while T_{TP} increase slightly. The dramatic increase of T_{Cent} illustrates that the overheads incurred by maintaining linkage structure and updating statistic information rise considerably when data size increases. Before the point $S = 250$ MB, T_{TP} is greater than T_{Cent}. But after the point, T_{TP} is greater than T_{Cent}, and the difference between T_{Cent} and T_{TP} becomes larger and larger with the size of data sets increasing. The trend of T_{TP} and T_{Cent} indicates that TPTDS becomes more efficient compared with CentTDS for large-scale data sets.

In our experiments, CentTDS fails due to insufficient memory when the size of data set is greater than 500 MB. Hence, CentTDS suffers from scalability problem for large-scale data sets. To further evaluate the scalability and efficiency of TPTDS, we run TPTDS over data sets with larger sizes. Fig. 2b shows the change of T_{TP} with respect to the data size ranging from 500 MB to 2.5 GB. It can be seen from Fig. 2b that T_{TP} grows linearly and stably with respect to the size of data sets. Based on the tendency of T_{TP}, we maintain that TPTDS is capable of scaling over large-scale data sets efficiently.

The above experimental results demonstrate that our approach can significantly improve the scalability and efficiency compared with the state-of-the-art TDS approach when anonymizing large-scale data sets.

In the last two groups, we explore the change of T_{TP} and IL_{TP} with respect to parameters k^I and p, respectively. We use the values T_{TP} and IL_{TP} of MRTDS as baselines to evaluate the effectiveness of TPTDS, since TPTDS makes trade-offs between scalability and data utility based on MRTDS. As MRTDS has the same data utility as CentTDS, the *ILoss* of MRTDS is the lower bound of *ILoss* for TPTDS. For convenience, the baseline execution time and *ILoss* are denoted as T_{Base} and IL_{Base}, respectively. The values of T_{Base} and IL_{Base} are calculated by setting $p = 1$. T_{Base} and IL_{Base} never vary with respect to k^I and p, but we plot them in figures for intuitive comparison. In both groups, we set the size of data sets as 500 MB which is large enough for evaluation according to the results in the first group.

In the second group, p is set as 3. The value of p ($p > 1$) is selected randomly and does not affect our analysis as what we want to see is the trend of T_{TP} and IL_{TP} with respect to k^I. Interesting readers can try other values. The conclusions will be the same. The results of this group are depicted in Fig. 3. As the number of records in the data set is around 5,000,000, k^I varies within [50, 5,000,000]. For conciseness, k^I is indicated by *Exp* which is the exponent of the scientific notation of k^I, i.e., $k^I = 5 * 10^{Exp}$. So, *Exp* ranges from 1 to 6. Fig. 3a shows the change of execution time T_{TP} against T_{Base} with respect to k^I, while Fig. 3b shows the change of IL_{TP} against IL_{Base} with respect to k^I.

From Fig. 3a, it is seen that T_{TP} is not greater than T_{Base} at each k^I value, meaning that TPTDS can be more efficient than the baseline. T_{TP} decreases with k^I increasing and hits the bottom at $Exp = 4$. Then, it goes up after $Exp = 4$ and arrives at the baseline. The change of T_{TP} illustrates high scalability and efficiency can be obtained by setting k^I as a neither too great nor too little value. As to *ILoss*, Fig. 3b presents that IL_{TP} is lightly greater than IL_{Base} within the overall domain, illustrating that splitting computation in two phases leads to higher *ILoss*. IL_{TP} is getting less in relation to the rise of k^I and reaches the baseline at $Exp = 6$. This trend of IL_{TP} reveals that the less computation is conducted in the first phase, the lower *ILoss* is incurred. At the point $Exp = 4$, T_{TP} reduces by nearly 50 percent compared to the baseline while the IL_{TP} only grows by less than 10 percent. In terms of this observation, TPTDS can gain higher efficiency than the baseline at the cost of slight increase of *ILoss*, via properly tuning parameter k^I.

In the third group, k^I is set as 50,000. The value is selected randomly and does not affect our analysis because what we want to see is the trend of T_{TP} and IL_{TP} with respect to the number of partitions. The number of partitions varies from 1 to 20. The results of this group are presented in Fig. 4. Fig. 4a demos the change of execution time T_{TP} against T_{Base} with respect to p, while Fig. 4b shows the change of IL_{TP} against IL_{Base} with respect to p.

Fig. 4a shows that T_{TP} varies a lot with respect to p. Before the point $p = 12$, T_{TP} is less than the baseline, meaning that parallelization of MapReduce jobs brings benefits. On the contrary, T_{TP} is greater than the baseline after the point, resulting in low efficiency. When $p < 16$, T_{TP} drops first and grows with the increase of p. In terms of the above observation, parallelization of multiple jobs also engenders overheads, and only proper degree of job level parallelization can advance scalability and efficiency. From Fig. 4b, IL_{TP} is not less than the baseline with the growth of p, showing that parallelization of multiple jobs can indeed engender higher *ILoss* as cost. IL_{TP} increases first and then drops subsequently when p is getting greater. When $p = 4$, IL_{TP} is nearly 50 percent of the baseline, while IL_{TP} only increase by less than 10 percent. Similar to k_I, TPTDS can gain higher efficiency than the baseline at the cost of slight increase of *ILoss*, via properly tuning parameter p.

The second and third groups of experiments show that TPTDS can gain high scalability at certain cost of data utility, and it can offer an optimized tradeoff between efficiency and data utility with tuning well-tuned parameters.

As a conclusion, all the experimental results demonstrate that our approach significantly improves the scalability and efficiency of TDS over existing TDS approaches.

7 Conclusions and Future Work

In this paper, we have investigated the scalability problem of large-scale data anonymization by TDS, and proposed a highly scalable two-phase TDS approach using MapReduce on cloud. Data sets are partitioned and anonymized in parallel in the first phase, producing intermediate results. Then, the intermediate results are merged and further anonymized to produce consistent k-anonymous data sets in the second phase. We have creatively applied MapReduce on cloud to data anonymization and deliberately designed a group of innovative MapReduce jobs to concretely accomplish the specialization computation in a highly scalable way. Experimental results on real-world data sets have demonstrated that with our approach, the scalability and efficiency of TDS are improved significantly over existing approaches.

In cloud environment, the privacy preservation for data analysis, share and mining is a challenging research issue due to increasingly larger volumes of data sets, thereby requiring intensive investigation. We will investigate the adoption of our approach to the bottom-up generalization algorithms for data anonymization. Based on the contributions herein, we plan to further explore the next step on scalable privacy preservation aware analysis and scheduling on large-scale data sets. Optimized balanced scheduling strategies are expected to be developed towards overall scalable privacy preservation aware data set scheduling.

References

[1] S. Chaudhuri, "What Next?: A Half-Dozen Data Management Research Goals for Big Data and the Cloud," *Proc. 31st Symp. Principles of Database Systems (PODS '12)*, pp. 1-4, 2012.

[2] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A View of Cloud Computing," *Comm. ACM*, vol. 53, no. 4, pp. 50-58, 2010.

[3] L. Wang, J. Zhan, W. Shi, and Y. Liang, "In Cloud, Can Scientific Communities Benefit from the Economies of Scale?," *IEEE Trans. Parallel and Distributed Systems*, vol. 23, no. 2, pp.296-303, Feb. 2012.

[4] H. Takabi, J.B.D. Joshi, and G. Ahn, "Security and Privacy Challenges in Cloud Computing Environments," *IEEE Security and Privacy*, vol. 8, no. 6, pp. 24-31, Nov. 2010.

[5] D. Zissis and D. Lekkas, "Addressing Cloud Computing Security Issues," *Future Generation Computer Systems*, vol. 28, no. 3, pp. 583-592, 2011.

[6] X. Zhang, C. Liu, S. Nepal, S. Pandey, and J. Chen, "A Privacy Leakage Upper-Bound Constraint Based Approach for Cost-Effective Privacy Preserving of Intermediate Data Sets in Cloud," *IEEE Trans. Parallel and Distributed Systems*, to be published, 2012.

[7] L. Hsiao-Ying and W.G. Tzeng, "A Secure Erasure Code-Based Cloud Storage System with Secure Data Forwarding," *IEEE Trans. Parallel and Distributed Systems*, vol. 23, no. 6, pp. 995-1003, 2012.

[8] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-Preserving Multi-Keyword Ranked Search over Encrypted Cloud Data," *Proc. IEEE INFOCOM*, pp. 829-837, 2011.

[9] P. Mohan, A. Thakurta, E. Shi, D. Song, and D. Culler, "Gupt: Privacy Preserving Data Analysis Made Easy," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '12)*, pp. 349-360, 2012.

[10] *Microsoft HealthVault*, http://www.microsoft.com/health/ww/products/Pages/healthvault.aspx, 2013.

[11] B.C.M. Fung, K. Wang, R. Chen, and P.S. Yu, "Privacy-Preserving Data Publishing: A Survey of Recent Devel- opments," *ACM Computing Surveys*, vol. 42, no. 4, pp. 1-53, 2010.

[12] B.C.M. Fung, K. Wang, and P.S. Yu, "Anonymizing Classification Data for Privacy Preservation," *IEEE Trans. Knowledge and Data Eng.*, vol. 19, no. 5, pp. 711-725, May 2007.

[13] X. Xiao and Y. Tao, "Anatomy: Simple and Effective Privacy Preservation," *Proc. 32nd Int'l Conf. Very Large Data Bases (VLDB '06)*, pp. 139-150, 2006.

[14] K. LeFevre, D.J. DeWitt, and R. Ramakrishnan, "Incognito: Efficient Full-Domain K-Anonymity," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '05)*, pp. 49-60, 2005.

[15] K. LeFevre, D.J. DeWitt, and R. Ramakrishnan, "Mondrian Multidimensional K-Anonymity," *Proc. 22nd Int'l Conf. Data Eng. (ICDE '06)*, 2006.

[16] V. Borkar, M.J. Carey, and C. Li, "Inside 'Big Data Management': Ogres, Onions, or Parfaits?," *Proc. 15th Int'l Conf. Extending Database Technology (EDBT '12)*, pp. 3-14, 2012.

[17] K. LeFevre, D.J. DeWitt, and R. Ramakrishnan, "Workload-Aware Anonymization Techniques for Large-Scale Data Sets," *ACM Trans. Database Systems*, vol. 33, no. 3, pp. 1-47, 2008.

[18] T. Iwuchukwu and J.F. Naughton, "K-Anonymization as Spatial Indexing: Toward Scalable and Incremental Anonymization," *Proc. 33rd Int'l Conf. Very Large Data Bases (VLDB '07)*, pp. 746-757, 2007.

[19] J. Dean and S. Ghemawat, "Mapreduce: Simplified Data Processing on Large Clusters," *Comm. ACM*, vol. 51, no. 1, pp. 107-113, 2008.

[20] N. Mohammed, B. Fung, P.C.K. Hung, and C.K. Lee, "Centralized and Distributed Anonymization for High-Dimensional Healthcare Data," *ACM Trans. Knowledge Discovery from Data*, vol. 4, no. 4, Article 18, 2010.

[21] B. Fung, K. Wang, L. Wang, and P.C.K. Hung, "Privacy-Preserving Data Publishing for Cluster Analysis," *Data and Knowledge Eng.*, vol. 68, no. 6, pp. 552-575, 2009.

[22] N. Mohammed, B.C. Fung, and M. Debbabi, "Anonymity Meets Game Theory: Secure Data Integration with Malicious Participants," *VLDB J.*, vol. 20, no. 4, pp. 567-588, 2011.

[23] L. Sweeney, "k-Anonymity: A Model for Protecting Privacy," *Int'l J. Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 10, no. 5, pp. 557-570, 2002.

[24] W. Jiang and C. Clifton, "A Secure Distributed Framework for Achieving k-Anonymity," *VLDB J.*, vol. 15, no. 4, pp. 316-333, 2006.

[25] P. Jurczyk and L. Xiong, "Distributed Anonymization: Achieving Privacy for Both Data Subjects and Data Providers," *Proc. 23rd Ann. IFIP WG 11.3 Working Conf. Data and Applications Security XXIII (DBSec '09)*, pp. 191-207, 2009.

[26] I. Roy, S.T.V. Setty, A. Kilzer, V. Shmatikov, and E. Witchel, "Airavat: Security and Privacy for Mapreduce," *Proc. Seventh USENIX Conf. Networked Systems Design and Implementation (NSDI '10)*, pp. 297-312, 2010.

[27] K. Zhang, X. Zhou, Y. Chen, X. Wang, and Y. Ruan, "Sedic: Privacy-Aware Data Intensive Computing on Hybrid Clouds," *Proc. 18th ACM Conf. Computer and Comm. Security (CCS '11)*, pp. 515-526, 2011.

[28] X. Xiao and Y. Tao, "Personalized Privacy Preservation," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '06)*, pp. 229-240, 2006.

[29] Amazon Web Services, "Amazon Elastic Mapreduce," http://aws.amazon.com/elasticmapreduce/, 2013.

[30] Apache, "Hadoop,"http://hadoop.apache.org, 2013.

[31] Y. Bu, B. Howe, M. Balazinska, and M.D. Ernst, "The Haloop Approach to Large-Scale Iterative Data Analysis," *VLDB J.*, vol. 21, no. 2, pp. 169-190, 2012.

[32] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox, "Twister: A Runtime for Iterative Mapreduce," *Proc. 19th ACM Int'l Symp. High Performance Distributed Computing (HDPC '10)*, pp. 810-818, 2010.

[33] KVM, http://www.linux-kvm.org/page/Main_Page, 2013.

[34] OpenStack, http://openstack.org/, 2013.

[35] UCI Machine Learning Repository, ftp://ftp.ics.uci.edu/pub/machine-learning-databases/, 2013.

Xuyun Zhang received the bachelor's and master's degrees in computer science from Nanjing University, China. He is currently working toward the PhD degree at the Faculty of Engineering and IT, University of Technology, Sydney, Australia. His research interests include cloud computing, privacy and security, Big Data, MapReduce, and OpenStack. He has published several papers in refereed international journals including *IEEE Transactions on Parallel and Distributed Systems (TPDS)*.

Laurence T. Yang received the BE degree in computer science and technology from Tsinghua University, China, and the PhD degree in computer science from the University of Victoria, Canada. He is a professor in the School of Computer Science and Technology, Huazhong University of Science and Technology, China and in the Department of Computer Science, St. Francis Xavier University, Canada. His current research interests include parallel and distributed computing, embedded and ubiquitous computing. His research has been supported by National Sciences and Engineering Research Council, Canada and Canada Foundation for Innovation. He is a senior member of the IEEE.

Chang Liu is currently working toward the PhD degree at the University of Technology Sydney, Australia. His research interests include cloud computing, resource management, cryptography, and data security.

Jinjun Chen received the PhD degree in computer science and software engineering from Swinburne University of Technology, Australia. He is an associate professor from the Faculty of Engineering and IT, University of Technology Sydney (UTS), Australia. He is the director of Lab of Cloud Computing and Distributed Systems at UTS. His research interests include cloud computing, Big Data, workflow management, privacy and security, and related various research topics. His research results have been published in more than 100 papers in high-quality journals and at conferences, including *IEEE Transactions on Service Computing*, *ACM Transactions on Autonomous and Adaptive Systems*, *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *IEEE Transactions on Software Engineering (TSE)*, and *IEEE Transactions on Parallel and Distributed Systems (TPDS)*. He received Swinburne Vice-Chancellor's Research Award for early career researchers (2008), IEEE Computer Society Outstanding Leadership Award (2008-2009) and (2010-2011), IEEE Computer Society Service Award (2007), Swinburne Faculty of ICT Research Thesis Excellence Award (2007). He is an associate editor for the *IEEE Transactions on Parallel and Distributed Systems*. He is the vice chair of IEEE Computer Society's Technical Committee on Scalable Computing (TCSC), vice chair of Steering Committee of Australasian Symposium on Parallel and Distributed Computing, founder and coordinator of IEEE TCSC Technical Area on Workflow Management in Scalable Computing Environments, founder and steering committee co-chair of International Conference on Cloud and Green Computing, and International Conference on Big Data and Distributed Systems. He is a member of the IEEE.