Journal of Computer and System Sciences 80 (2014) 1008–1020

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

A hybrid approach for scalable sub-tree anonymization over big data using MapReduce on cloud

Xuyun Zhang [a,*], Chang Liu [a], Surya Nepal [b], Chi Yang [a], Wanchun Dou [c], Jinjun Chen [a]

[a] Faculty of Engineering and Information Technology, University of Technology Sydney, PO Box 123, Broadway, NSW 2007, Australia
[b] Centre for Information & Communication Technologies, Commonwealth Scientific and Industrial Research Organisation, Cnr Vimiera and Pembroke Roads, Marsfield, NSW 2122, Australia
[c] State Key Laboratory for Novel Software Technology, Nanjing University, 22 Hankou Road, Nanjing, Jiangsu Province 210093, China

ARTICLE INFO

Article history:
Received 25 September 2012
Received in revised form 15 March 2013
Accepted 27 August 2013
Available online 11 February 2014

Keywords:
Big data
Cloud computing
Data anonymization
Privacy preservation
MapReduce

ABSTRACT

In big data applications, data privacy is one of the most concerned issues because processing large-scale privacy-sensitive data sets often requires computation resources provisioned by public cloud services. Sub-tree data anonymization is a widely adopted scheme to anonymize data sets for privacy preservation. Top–Down Specialization (TDS) and Bottom–Up Generalization (BUG) are two ways to fulfill sub-tree anonymization. However, existing approaches for sub-tree anonymization fall short of parallelization capability, thereby lacking scalability in handling big data in cloud. Still, either TDS or BUG individually suffers from poor performance for certain valuing of k-anonymity parameter. In this paper, we propose a hybrid approach that combines TDS and BUG together for efficient sub-tree anonymization over big data. Further, we design MapReduce algorithms for the two components (TDS and BUG) to gain high scalability. Experiment evaluation demonstrates that the hybrid approach significantly improves the scalability and efficiency of sub-tree anonymization scheme over existing approaches.

1. Introduction

Cloud computing and big data are two disruptive trends at present, imposing significant impacts on current IT industry and research communities [1,2]. Cloud computing elastically provides massive computation power and storage capacity via utilizing a large number of commodity computers together, enabling users to deploy big data applications cost-effectively without heavy infrastructure investment. To exploit the advantages offered by public cloud platforms, more and more big data applications have been moving into cloud, including a variety of privacy-sensitive applications like health care data and transactional data storing and processing. Consequently, how to protect the privacy of these data sets turns out to be a big challenge. Indeed, numerous potential customers are still hesitant to take advantage of cloud due to privacy and security concerns [3]. Privacy is one of the most concerned issues in the big data applications that involve multiple parties, and the concern aggravates in the context of cloud computing although some privacy issues are not new [1]. Hence, data privacy issues need to be addressed urgently before data sets are analyzed or shared on cloud.

* Corresponding author.
E-mail addresses: xyzhanggz@gmail.com (X. Zhang), changliu.it@gmail.com (C. Liu), Surya.Nepal@csiro.au (S. Nepal), douwc@nju.edu.cn (W. Dou), jinjun.chen@gmail.com (J. Chen).

http://dx.doi.org/10.1016/j.jcss.2014.02.007
0022-0000/

Data anonymization, extensively studied and widely adopted [4], is an effective way for data privacy preservation. 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. Sub-tree anonymization scheme is widely adopted to anonymize data sets for privacy preservation, producing a good trade-off between data utility and distortion. There are two ways to accomplish sub-tree anonymization, i.e., Top–Down Specialization (TDS) and Bottom–Up Generalization (BUG). So far, a series of approaches have been proposed for TDS or BUG [5–8]. However, data sets in big data applications on cloud have become so large that it is a big challenge for existing sub-tree anonymization algorithms to anonymize such data sets in a scalable fashion, due to their lack of parallelization capability. Still, either TDS or BUG individually suffers from poor performance for certain valuing of k-anonymity parameter, if k-anonymity privacy model [9] is adopted. Specifically, TDS is preferred when k is large while BUG is favorable when k is small.

MapReduce [10], a large-scale data processing framework, have been integrated with cloud to provide powerful computation capability for applications, e.g. Amazon Elastic MapReduce (EMR) service [11]. We leverage MapReduce to address the scalability problem in our approach. As the MapReduce computation paradigm is relatively simple, it is still a challenge to design proper MapReduce jobs for TDS and BUG.

In this paper, we propose a highly scalable hybrid approach that combines TDS and BUG together for sub-tree anonymization over big data. The approach automatically determines which component is used to conduct the anonymization when a data set is given, by comparing the user-specified k-anonymity parameter with a threshold derived from the data set. Both components TDS and BUG are developed based on MapReduce to gain high scalability by exploiting powerful computation capability of cloud. Since having designed MapReduce based TDS in [12], we only present the MapReduce algorithmic design of BUG herein. Experimental evaluation demonstrates that the hybrid approach significantly improves the scalability and efficiency of sub-tree data anonymization over existing approaches.

The major contributions of our search are four fold. Firstly, we propose a hybrid approach to improve the scalability and efficiency of sub-tree data anonymization via automatically choosing TDS or BUG. Secondly, a group of innovative MapReduce jobs are designed for BUG to concretely conduct the computation in a highly scalable fashion. Thirdly, we propose to perform multiple generalization operations in one round of iteration, boosting the parallelization capability and scalability of BUG. Lastly, experimental evaluation demonstrates that the hybrid approach improves the scalability and efficiency of sub-tree anonymization scheme over existing approaches.

The remainder of this paper is organized as follows. The next section reviews related work, and analyzes the problems in existing sub-tree anonymization approaches. In Section 3, we briefly present the preliminary for our approach. Section 4 elaborates algorithmic details of MapReduce jobs for BUG, and Section 5 formulates the hybrid approach. We empirically evaluate our approach in Section 6. Finally, we conclude this paper and discuss future work in Section 7.

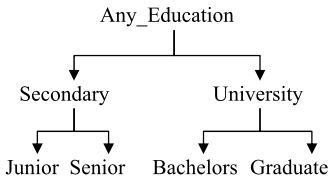
2. Related work and problem analysis

2.1. Related work

Privacy preservation on data has been extensively investigated and fruitful progress has been made by research communities [4]. We briefly review the related work as follows.

A bulk of privacy models and anonymization approaches have been put forth to preserve the privacy-sensitive information in data sets. k-anonymity [9] and l-diversity [13] are two basic and widely-adopted privacy models to measure the degree of privacy-sensitive information disclosure against record linkage attacks and attribute linkage attacks, respectively. Other privacy models like t-closeness [14] and m-invariance [15] are also proposed for various privacy attack scenarios. Several anonymizing operations are leveraged to anonymize data sets, including suppression [16], generalization [5,17,18], anatomization [19], slicing [20], disassociation [21], etc. In this paper, we utilize generalization to anonymize data sets. A generalization replaces some domain values with a parent value in the taxonomy of an attribute. Roughly, there are four generalization schemes listed as follows [4]. Full-domain generalization [22] makes all domain values in an attribute generalized to the same level of the taxonomy tree. Sub-tree generalization [5] requires that either all child domain values or none of a non-leaf node in a taxonomy tree are generalized. Multidimensional generalization [17] takes multiple attributes together into account when generalizing domain values. The aforementioned schemes are global recoding, i.e., if a domain value is generalized, all its instances are generalized. Cell generalization [18], on the contrary, is local recoding which generalizes identical instances into different levels of domain values.

Our research herein concentrates on the sub-tree generalization scheme. Unlike multidimensional or cell generalization schemes, sub-tree generalization can produce consistent anonymous data that can be directly used by existing data mining and data analysis tools. This scheme offers a good trade-off between data utility and data consistency. Thus, this scheme has been extensively explored. Top–down specialization [5–7,12,23] and bottom–up generalization [8] are two categories of methods to fulfill the sub-tree generalization scheme. Most exiting algorithms exploit indexing data structure to assist the process of anonymization. Specifically, TIPS (Taxonomy Indexed PartitionS) for top–down specialization and TEA (Taxonomy Encoded Anonymity) index for bottom–up generalization. Although indexing data structures can speed up the process of data anonymization, these approaches often fail to work in parallel or distributed environments like cloud



Education	# of Recs.
Junior	2
Senior	2
Bachelors	3
Graduate	3

Fig. 1. Education taxonomy tree and records.

systems because the indexing structures are centralized. Mohammed et al. [6] proposed a distributed top–down specialization approach which, however, mainly concerns privacy protection against other parities rather than scalability issues. Still, this approach only employs information gains as the search metric, resulting lower data utility than centralized ones. Our previous work [12] leverages MapReduce to accomplish the intensive computation required in big data anonymization via top–down specialization. But top–down specialization probably performs slower than bottom–up generalization when k-anonymity parameter k is small.

Scalability and efficiency of anonymization algorithms for privacy preservation has drawn attention of researchers. R-tree indexing, scalable decision trees and sampling techniques are introduced to achieve high scalability and efficiency [24,25]. However, the proposed approaches aim at multidimensional generalization scheme, thereby failing to work for sub-tree generalization. MapReduce has been widely adopted in various data processing applications to boost scalability and efficiency [26–28]. Following this line, we also leverage MapReduce to advance scalability and efficiency in our research on big data anonymization.

2.2. Problem analysis

In this section, we analyze the problem of utilizing Top–Down Specialization (TDS) or Bottom–Up Generalization (BUG) alone for sub-tree generalization, and the scalability problem of existing BUG approaches.

At present, existing TDS and BUG approaches are developed individually for sub-tree generalization scheme. Both of them are lack of the awareness of the user-specified k-anonymity parameter. In fact, the values of the k-anonymity parameter can impact their performance. Intuitively, if parameter k is large, TDS is more suitable while BUG will probably get bad performance. The case is reversed when k is small. A simple example is described below to demonstrate the above intuition.

Assume that a data set has 10 records with the attribute *Education* that needs generalization for privacy preservation. The taxonomy tree of this attribute and attribute value of records are depicted in Fig. 1. At one extreme, let k be set as 2. In this case, TDS has to specialize several domain values to achieve 2-anonymity while BUG will do nothing as the data set is already 2-anonymity. At other extreme, let k be set as 5. In such a case, TDS will do nothing to achieve 5-anonymity while BUG has to conduct generalization. It can be seen from the example that selecting TDS or BUG according to k significantly impacts the performance of the sub-tree anonymization scheme.

As such, it is promising to develop a hybrid approach that encompasses TDS and BUG as two components so that the high scalability and efficiency can be gained regardless of valuing of the k-anonymity parameter. The key to the hybrid approach is how to design a user-friendly method to automatically determine which component should be chosen. Although a domain expert is able to manually determine which approach is preferred to conduct anonymization according to the value of parameter k, ordinary users in the cloud environment probably fail to do this due to their lack of background knowledge.

As to BUG, the existing approach in [8] exploits indexing data structure to promote efficiency, thereby falling short of high scalability and parallelization in cloud environments. Thus, it is worthwhile investigating how to develop BUG algorithm with MapReduce in order to improve the scalability and efficiency. A promising way is to conduct generalization operations in parallel. As MapReduce only provides primitive programming model, how to develop MapReduce jobs to conduct the computation required by data anonymization is still critical in the whole design and needs intensive research.

Above all, the problem we attempt to address is how to design scalable and efficient BUG algorithm based on MapReduce and how to automatically select a component for the proposed hybrid approach according to parameter k.

3. Preliminary

3.1. Sub-tree generalization scheme

To facilitate subsequent discussion, we briefly introduce the sub-tree generalization scheme as background knowledge. Table 1 lists some basic symbols and notations. We assume there is one sensitive value in one record like medical diagnosis, as more sensitive values will not affect our discussion. The leaf node values of a taxonomy tree TT_i are original attribute values of records in D. We use QI as the acronym of quasi-identifier. The quasi-identifier group is abbreviated as QI-group [29]. Without loss of generality, we adopt k-anonymity [9] as the privacy model in this paper, i.e., for any $qid \in QID$, the size of $QJG(qid)$ must be zero or at least k, so that a quasi-identifier will not be distinguished from other at least $k-1$ quasi-identifiers.

In sub-tree generalization scheme, all child values of a non-leaf node or none in a domain hierarchy are generalized to the node. Two operations can be employed to accomplish this scheme, i.e., generalization for BUG and specialization for TDS,

Table 1
Basic symbols and notations.

Symbol	Definition
D	a data set containing data records
m	number of attributes
r	a data record, $r \in D$ and $r = \langle v_1, v_2, \ldots, v_m, sv \rangle$, where v_i, $1 \leqslant i \leqslant m$, is an attribute value, and sv is a sensitive value
$Attr_i$	the ith attribute of a data record
TT_i	the taxonomy tree of the attribute $Attr_i$
DOM_i	the set of all domain values in TT_i
SV	the set sensitive values
qid	a quasi-identifier, $qid = \langle q_1, q_2, \ldots, q_m \rangle$, $q_i \in DOM_i$, $1 \leqslant i \leqslant m$
QID	the set of quasi-identifiers, $QID = \langle Attr_1, Attr_2, \ldots, Attr_m \rangle$
$QIG(qid)$	quasi-identifier group containing all records with the same quasi-identifier qid
$AGSet$	available generalization set
$SGSet$	sibling generalization set
$NGSet$	new generalization set
$AQISet$	anonymity quasi-identifier set
$CGSet$	critical generalization set
$NCGSet$	non-critical generalization set
$RGSet$	racing generalization set
$ACGSet$	available critical generalization set

respectively. A generalization operation is to replace a domain value with its parent in a taxonomy tree while a specialization operation is to replace a domain value with its all child values. Formally, a generalization is represented as $gen : Child(q) \rightarrow q$ while a specialization is represented as $spec : q \rightarrow Child(q)$, where $q \in DOM_i$ is a domain value and the set $Child(q)$ consists of all child domain values of q. We leverage the concept anonymization level [12] to capture the degree of anonymization. Specifically, anonymization level, denoted as AL, is a vector of domain values sets, i.e., $AL = \langle Cut_1, Cut_2, \ldots, Cut_m \rangle$, where Cut_i, $1 \leqslant i \leqslant m$, is the cut of TT_i. A cut of a tree is a subset of values in the tree that contains exactly one value on each root-to-leaf path [4].

To guide the selection of the best operations in the anonymization process, the goodness of a candidate generalization or specialization is measured by a search metric. We leverage the information/privacy trade-off as the search metric for our approach, i.e., the Information Gain per Privacy Loss (IGPL) for TDS and the Information Loss per Privacy Gain (ILPG) for BUG, respectively [4]. We briefly describe how to calculate the value of ILPG subsequently. Interested readers can refer to [12] for IGPL calculation or [4] for more details.

Given a generalization $gen : Child(q) \rightarrow q$, the ILPG of the generalization is calculated by

$$ILPG(gen) = IL(gen) / \big(PG(gen) + 1\big). \tag{1}$$

The term $IL(gen)$ is the information loss after performing gen, and $PG(gen)$ is the privacy gain. Both of them are 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, $IL(gen)$ is given by

$$IL(gen) = \sum_{c \in Child(q)} \left(\frac{|R_c|}{|R_q|} \right) I(R_c) - I(R_q). \tag{2}$$

Let $A_p(gen)$ denote the anonymity after performing gen, while $A_c(gen)$ be that before performing gen. Then, the privacy gain from gen is calculated by

$$PG(gen) = A_p(gen) - A_c(gen). \tag{3}$$

3.2. MapReduce basics

MapReduce is a scalable and fault-tolerant data processing framework that is capable of processing huge volume of data in parallel with many low-end commodity computers [10]. It has been widely adopted and received extensive attention from both academia and industry due to its promising capability. Simplicity, scalability and fault-tolerance are three main salient features of the MapReduce framework. In general, a MapReduce job consists of two primitive functions, *Map* and *Reduce*, defined over a data structure named key-value pair $(key, value)$. Specifically, the *Map* function can be formalized as $Map : (k_1, v_1) \rightarrow (k_2, v_2)$, i.e., *Map* takes a pair (k_1, v_1) as input and then outputs another intermediate key-value pair (k_2, v_2). These intermediate pairs are consumed by the *Reduce* function as input. Formally, the *Reduce* function can be represented as $Reduce : (k_2, list(v_2)) \rightarrow (k_3, v_3)$, i.e., *Reduce* takes intermediate k_2 and all its corresponding values $list(v_2)$ as input and outputs another pair (k_3, v_3). Usually, (k_3, v_3) list is the results which MapReduce users attempt to obtain. Both *Map* and *Reduce* functions are specified by users according to their specific applications. An instance running *Map* function is called *Mapper*, and that running *Reduce* function is called *Reducer*, respectively.

To make such a simple programming model powerful, MapReduce provides many fundamental mechanisms such as data replication and data sorting. Between *Map* phase and *Reduce* phase exists a *Shuffle* phase, during which the intermediate

key-value pairs are sorted according to keys. This mechanism is quite useful for many complex applications to improve scalability. Moreover, distributed file systems like HDFS (Hadoop Distributed File System) [30] are substantially crucial to make MapReduce framework run in a highly scalable and fault-tolerant fashion.

4. Bottom–up generalization using MapReduce

We mainly elaborate Bottom–Up Generalization using MapReduce (MRBUG) in this section. Basically, a practical MapReduce program encompasses *Map* and *Reduce* functions, and a Driver that coordinates the macro execution of MapReduce jobs. Thus, we describe the Bottom–Up Generalization MapReduce Driver in Section 4.1 to present the basic process of bottom–up generalization. To improve the scalability and efficiency of our approach, parallelization degree of bottom–up generalization is boosted in Section 4.2. Sections 4.3 and 4.4 present the MapReduce jobs in detail.

4.1. Bottom–up generalization MapReduce driver

Basically, Bottom–Up Generalization (BUG) approach of anonymization is an iterative process starting from the lowest anonymization level. The lowest anonymization level contains the internal domain nodes in the lowest level of taxonomy trees. Each round of iteration includes four major steps, namely, checking the current data set whether satisfies the anonymity requirement, calculating the ILPG, finding the best generalization and generalizing the data set according to the selected best generalization. Calculating the ILPG and generalizing the data set involve accessing a large number of data records, thereby dominating the scalability and efficiency of bottom–up generalization. An existing approach [8] utilizes indexing data structure and retaining statistic information to improve the efficiency. But the approach suffers from poor scalability and efficiency in big data scenario. Still, the approach fails to be adapted into MapReduce since MapReduce does not support indexing data structure.

As such, we propose to develop innovative MapReduce jobs for the ILPG computation. As the notion of anonymization level is introduced to describe anonymization status of a data set, it is unnecessary to generalize the data set concretely in each round in regards to efficiency. Instead, we abstractly generalize the data set over the current anonymization level. After the final anonymization level is obtained, we anonymize the data set in a one-pass MapReduce job. Algorithm 1 presents the Bottom–Up Generalization MapReduce (BUGMR) driver.

Algorithm 1 MRBUG driver.

Input: Data set D, anonymization level AL_0, anonymity parameter k.
Output: Final anonymous data set D^*.

1: Initialize the values of search metric ILPG for each generalization $gen \in \bigcup_{j=1}^{m} Cut_j$ with respect to AL_0, via job *ILPG Calculation*;
2: while $\exists gen,\ A_c(gen) < k$
3: Identify the available generalization set *AGSet* out of all the active generalization candidates;
4: $\forall gen \in AGset$, label gen as *INACTIVE* to perform gen on the current anonymization level;
5: if $\exists gen \in AGset$, $\forall gen' \in SGSet(gen)$, gen' is already labeled as *INACTIVE*;
6: Insert a new generalization $gen_{New} : Child(q) \rightarrow q$ into *NGSet*, where $Child(q) = \{q_i \mid gen' : Child(q_i) \rightarrow q_i,\ gen' \in SGSet(gen)\}$;
7: Remove all generalizations in $SGSet(gen)$;
8: end if
9: $AL_{i+1} \leftarrow AL_i$; Update ILPG values for all active generalization candidates in AL_{i+1} via *ILPG Calculation*;
10: end while
11: Generalize D to D^* in terms of AL_i, via job *Data Generalization*.

We detail Algorithm 1 as follows. Firstly, ILPG values of all generalizations are initialized (line 1). Line 2 checks whether the current anonymized data set satisfies the k-anonymity requirement. Line 3 identifies the available generalization set which is denoted as *AGSet*. Basically, *AGSet* contains the best generalization gen_{Best} with the highest ILPG value only in terms of the third step of BUG approach mentioned above. But we perform multiple generalizations in one round of iteration in MRBUG in order to improve the degree of parallelization and efficiency, which will be elaborated in Section 4.2. Line 4 performs the generalizations in *AGSet* by labeling them as *INACTIVE*. That a generalization is labeled as *INACTIVE* means the generalization will not be considered any more in the following rounds, abstractly fulfilling anonymization on the data set. Let $SGSet(gen)$ denote the set containing generalization gen and its all siblings in the domain taxonomy tree. When the generalizations in $SGSet(gen)$ are all labeled as *INACTIVE*, a new higher level generalization will be inserted into the anonymization level to replace these inactive ones (lines 5, 6 and 7). Note that this is a remarkable difference from TDS. Since multiple generalizations in *AGSet* are checked for this, it is probably that more than one new generalizations produced. Let *NGSet* be the set of such generalizations. So, line 6 adds new generalizations to *NGSet*. Line 9 updates the privacy gain of each active generalization because the performing generalizations in *AGSet* probably changes the anonymity of the data set. Also, information loss computation is required if new generalizations have been inserted, i.e., $NGSet \neq \emptyset$. As the last step, line 11 concretely anonymizes the data set according to the final anonymization level.

Lines 1 and 9 require ILPG Calculation that involves accessing to the original data set and computing statistic information over the data set. Line 11 also needs processing the whole data set. We leverage MapReduce to conduct the intensive computation in these situations. Specifically, we design a couple of innovative MapReduce jobs: the job ILPG Calculation for accomplishing the computation required in lines 1 and 9, and the job Data Generalization for achieving the final concrete

anonymization in line 11. The ILPG related MapReduce job is elaborated in Section 4.3, and job Data Generalization is in Section 4.4, respectively. In the following section, we discuss how to boost the parallelization of performing generalization in each round of iteration, for the sake of improving scalability and efficiency of BUG.

4.2. Parallelization of performing generalizations

Several observations probably help to design efficient MapReduce jobs for the ILPG Calculation. One is that, unlike top–down specialization that inserts several new specialization candidates into the current anonymization level in each round, bottom–up generalization only inserts a new generalization candidate after several rounds of generalization. Another is that conducting a generalization will not affect the information loss of another generalization candidate in terms of (2). Based on such observations, we can take multiple generalization candidates into account in one round, thereby improving the degree of parallelization and the efficiency of our approach. However, performing a generalization possibly changes the anonymity of the data set and privacy gain of each generalization candidate will be affected. To identify which candidates can be considered simultaneously in one round of iteration, we give the following definitions.

Definition 1 *(Anonymity quasi-identifier).* A quasi-identifier qid is an anonymity quasi-identifier if $|QIG(qid)| = \min_{qid' \in QID} |QIG(qid')|$, where $|\cdot|$ represents the size of a QI-group.

The size of a QI-group with an anonymity quasi-identifier is the anonymity of the data set according to Definition 1. Note that there are probably more than one anonymity quasi-identifiers on an anonymization level. Let *AQISet* denote the set of anonymity quasi-identifiers. Then, we define the term critical generalization as follows.

Definition 2 *(Critical generalization).* A generalization $gen : Child(q) \to q$, is said to be critical if $\exists q' \in Child(q)$, $q' \in \bigcup_{qid \in AQISet} \{q_i \mid q_i = Proj_i(qid),\ 1 \leqslant i \leqslant m\}$, where $Proj_i(qid)$ is the ith coordinate of qid.

Let *CGSet* denote the set of critical generalizations, i.e., $CGSet = \bigcup_{qid \in AQISet} \{q_i \mid q_i = Proj_i(qid),\ 1 \leqslant i \leqslant m\}$. The set of non-critical generalizations is represented as *NCGSet*, and $NCGSet = (\bigcup_{i=1}^{m} Cut_i)/CGSet$. Based on the above definitions, we have the following observation.

Observation 1. If a generalization $gen \in CGSet$, performing gen possibly changes the anonymity of the data set, i.e., $A_p(gen) - A_c(gen)$ be possibly greater than 0. On the contrary, if $gen \in NCGSet$, $A_p(gen) - A_c(gen) = 0$, i.e., $PG(gen) = 0$.

Assume all the generalization candidates are sorted ascendingly according to the ILPG values. In terms of Observation 1, we can conduct all the candidates before the first critical generalization simultaneously, without affecting the anonymization result. To identify which critical generalizations can be generalized in the same round of iteration, the following definition is given.

Definition 3 *(Racing generalizations).* Assume all the generalization candidates are sorted ascendingly according to the ILPG values. The first critical generalization and its continuously succeeding critical generalizations are racing generalizations. They together constitute a set named racing generalization set, denoted as *RGSet*.

The first critical generalization can be performed in the same round definitely while others in *RGSet* possibly do this. Note that conducting critical generalizations potentially affects the ILPG values of candidates, thereby updating ILPG values is mandatory. To identify the available critical generalizations in one round from *RGSet*, a subroutine is presented in Algorithm 2. Let *ACGSet* denote the resultant available critical generalization set, i.e., the generalizations in *ACGSet* can be performed in one round of iteration together with the generalization before the first critical generalization. In the algorithm, a priority queue is leveraged to keep the generalizations sorted ascendingly with respect to ILPG.

Algorithm 2 Identifying available generalizations.

Input: *RGSet*, *AQISet*.
Output: *ACGSet*.
1: $Queue \leftarrow \emptyset$, where *Queue* is a priority queue in regard to ILPG;
2: while $RGSet \neq \emptyset$
3: $Queue \leftarrow g_{min}$, $g_{min} \in RGSet$ and g_{min} has the lowest ILPG;
4: $ACGSet \leftarrow g_{min}$;
5: while $Queue \neq \emptyset$
6: $g \leftarrow Queue$ and $RGSet \leftarrow RGSet/\{g\}$;
7: $Queue \leftarrow \bigcup_{qid \in \{qid' \mid qid' \text{ contains } g \text{ and } qid' \in AQISet\}} \{q_i \mid q_i = Proj_i(qid),\ 1 \leqslant i \leqslant m\}/\{g\}$;
8: $AQISet \leftarrow AQISet/\{qid' \mid qid' \text{ contains } g\}$;
9: end while
10: end while

Input parameter *RGSet* can be obtained readily after sorting all the active generalization candidates and identifying critical generalizations. According to Definition 2, identifying *CGSet* relies on the other input parameter of Algorithm 2, i.e., *AQISet*. Hence, identifying *AQISet* is a key to Algorithm 2. In Section 4.3, we will show how to identify *AQISet* in the MapReduce job ILPG Calculation. Once *ACGSet* is identified, we can construct available generalization set *AGSet* with ease, i.e., $AGSet = ACGSet \cup \{gen \mid gen$ locates before the first critical generalization$\}$.

4.3. ILPG Calculation job

The ILPG Calculation job is responsible to ILPG initialization in line 1 of Algorithm 1 and ILPG update in line 9. The computation required in ILPG initialization is quite similar to that of ILPG update. The *Map* function of the ILPG Calculation is depicted in Algorithm 3, while the *Reduce* function is presented in Algorithm 4. In Algorithms 3 and 4, the symbol '#' is used to identify whether a key is emitted to compute information gain or anonymity loss, and '$' is to differentiate the cases whether a key is for computing $A_p(spec)$ or $A_c(spec)$.

Algorithm 3 ILPG Calculation *Map*.

Input: Data record (ID_r, r), $r \in D$; anonymization level *AL*, *NGSet*.
Output: Intermediate key-value pair $(key, count)$.
1: For each attribute value v_i in r, find its generalization in current *AL*: gen_i. Let p_i be the parent in gen_i, and c_i be v_i itself or p'_i child that is also v'_i ancestor;
2: If $gen_i \in NGSet$, emit $(\langle p_i, c_i, sv\rangle, count)$;
3: Construct quasi-identifier $qid = \langle q_1, q_2, \ldots, q_m\rangle$, where $q_i = \begin{cases} p_i, & \text{if } gen_i \text{ is } INACTIVE, \\ c_i, & \text{o.w.}, \end{cases}$ $1 \leqslant i \leqslant m$; Emit $(\langle qid, \$, \#\rangle, count)$;
4: For each $i \in [1, m]$, replace q_i in qid with its parent p_i if $q_i = c_i$, producing the resultant quasi-identifier qid^*; emit $(\langle qid^*, p_i, \#\rangle, count)$.

Algorithm 3 is detailed as follows. For ILPG initialization, *NGSet* is the set of all the initial generalizations with respect to AL_0, while for ILPG updates, it is set according to line 6 of Algorithm 1. Line 1 of Algorithm 3 transforms an original record into its anonymized form according to the current anonymization level, for the sake of being counted. To compute $|R_p|$, $|(R_p, sv)|$, $|R_c|$ and $|(R_c, sv)|$ in (2) for information loss calculation, line 2 emits the key-value pair to the *Reduce* function for information loss computation if this pair is a new generalization candidate. Note that the information loss of a generalization will not be affected when we perform other generalizations or insert a new generalization, while privacy gain will probably be impacted as the anonymity of the data set will change.

Line 3 of Algorithm 3 aims at figuring out the anonymity of the data set before performing a generalization, i.e., $A_c(gen)$, while line 4 emits key-value pairs to obtain the anonymity after performing a generalization, i.e., $A_p(gen)$. As $A_c(gen)$ is unique globally, we just emit the current quasi-identifier *qid* for statistics. As to $A_p(gen)$, potential anonymous quasi-identifiers for *qid* will be emitted for computing $A_p(gen)$ for different active generalization candidates. After obtaining $A_p(gen)$ and $A_c(gen)$, we can update privacy gain for each generalization in terms of (3).

Algorithm 4 ILPG Calculation *Reduce*.

Input: Intermediate key-value pair $(key, list(count))$.
Output: Information gain $(gen, IL(gen))$ and anonymity $(gen, \langle A_c(gen), AQISet\rangle)$, $(gen, A_p(gen))$ for generalizations.
1: For each *key*, $sum \leftarrow \sum count$;
2: For each *key*, if $key.sv \neq \#$, update statistical counts:
3: $|(R_c, sv)| \leftarrow sum$, $|R_c| \leftarrow sum + |R_c|$, $|(R_p, sv)| \leftarrow sum + |(R_p, sv)|$, $|R_p| \leftarrow sum + |R_p|$;
4: If all sensitive values for child c have arrived, compute $I(R_c)$;
5: If all children c of parent p have arrived, compute $I(R_p)$ and $IL(gen)$; emit $(gen, IL(gen))$;
6: For each *key*, if $key.sv = \#$, update anonymity:
7: If $key.c = \$$ and $sum < A_p(gen)$, update current anonymity: $A_p(gen) \leftarrow sum$;
8: If $key.c \neq \$$:
9: If $sum < A_c(gen)$, update potential anonymity of $gen : A_c(gen) \leftarrow sum$ and $AQISet \leftarrow key.c$;
10: If $sum = A_c(gen)$, update: $AQISet \leftarrow \{key.c\} \cup AQISet$;
11: Emit $(gen, A_p(gen))$ and emit $(gen, \langle A_c(gen), AQISet\rangle)$.

The *Reduce* function described in Algorithm 4 mainly aggregates the statistical information to calculate information loss and privacy gain. The first part, lines 1 to 5, calculates information loss in terms of (2). Due to that the key-value pairs are sorted by MapReduce built-in mechanism before being fed to *Reducer* workers, the *Reduce* function can compute information loss for generalizations in sequence, without requiring large amount of memory to retaining statistical information. Therefore, the *Reduce* function is highly scalable for calculating information loss.

The essential of computing anonymity of a data set is to find out the minimum QI-group size. The second part, lines 6 to 11, aims at calculating privacy gain as well as identifying *AQISet*. The *Reducer* workers find out the locally minimum QI-group size before and after performing a generalization in parallel. Then, we can obtain the globally one in the driver program through comparing the outputs of *Reducer* workers. The quasi-identifiers of the QI-groups with the minimum group size are recorded during the process and constitute *AQISet*. Note that *AQISet* plays an important role in identifying available generalizations in the next round of iteration. Above all, the ILPG Calculation *Reduce* function is highly scalable for both

information loss and privacy gain computation. After obtaining information loss and privacy gain, we can calculate ILPG values according to (1).

4.4. Data Generalization

The original data set is concretely generalization for data anonymization by a one-pass MapReduce job, i.e., Data Generalization. Details of *Map* and *Reduce* functions of the data specialization MapReduce job are described in Algorithm 5. The *Map* function emits anonymous records and its count according to the current anonymization level. The *Reduce* function simply aggregates these anonymous records and counts their number. An anonymous record and its count represent a QI-group, and the QI-groups constitute the final anonymous data sets.

Algorithm 5 Data Generalization *Map* and *Reduce*.

Input: Data record (ID_r, r), $r \in D$; final anonymization level AL^*.
Output: Anonymous record $(r^*, count)$.
Map:
1: For each attribute value v_i in r, find its generalization in current AL: gen_i. Let p_i be the parent in gen_i, and c_i be v_i itself or p'_i child that is also v'_i ancestor;
2: Construct quasi-identifier $r^* = \langle q_1, q_2, \ldots, q_m \rangle$, where $q_i = \begin{cases} p_i, & \text{if } gen_i \text{ is } INACTIVE, \\ c_i, & \text{o.w.}, \end{cases}$ $1 \leqslant i \leqslant m$; emit $(r^*, count)$.
Reduce: For each r^*, $sum \leftarrow \sum count$; emit (r^*, sum).

5. Hybrid approach for sub-tree anonymization

In this section, we mainly discuss how to combine the top–down specialization and the bottom–up generalization together in the proposed hybrid approach. Section 5.1 sketches the hybrid approach which chooses components automatically by comparing the k-anonymity requirement with a system parameter. Then, we elaborate how to estimate the parameter in Section 5.2. Section 5.3 adjusts the parameter according to the data skewness to improve the performance of our approach.

5.1. Combining top–down specialization and bottom–up generalization

Now that the MapReduce version of Bottom–Up Generalization (MRBUG) has been developed in the last section, the two components, i.e., MRTDS [12] and MRBUG, are ready for the proposed hybrid approach of sub-tree anonymization over big data. In terms of the problem analysis in Section 2.2, we need to determine which component is used to anonymize data after the anonymity parameter k is specified by a user. It is promising that the hybrid approach can automatically give out a system parameter K such that if $k \geqslant K$, MRTDS is selected, otherwise MRBUG is selected. Formally, we define this threshold as Workload Balancing Point.

Definition 4 *(Workload balancing point).* An anonymity value of a data set, denoted as K, is defined as workload balancing point if it satisfies the condition that the amount of computation of anonymizing the data set to K-anonymous required by MRTDS is equal to that by MRBUG.

Once the workload balancing point K is identified, the hybrid approach can determine which component is chosen. If $k > K$, MRTDS is selected because MRBUG will incur more computation, while MRBUG is selected if $k < K$. The hybrid approach can be sketched in Algorithm 6 as follows.

Algorithm 6 Hybrid approach.

Input: Data set D; k-anonymity parameter k.
Output: Anonymous data set.
1: if $k \geqslant K$, then, anonymize D with MRTDS;
2: else, anonymize D with MRBUG.

Figuring out the exact value of K is difficult since K heavily depends on some properties of data sets, like data distribution and skewness. However, it is unnecessary to get the exact value because the performance of MRTDS and MRBUG just make a little difference when k is valued around K. As such, we roughly estimate the value k according to the size of the data set and taxonomy trees in the following section.

5.2. Workload balancing point estimation

To roughly estimate the workload balancing point, we assume that the values of an attribute are evenly distributed. Our basic idea is to estimate K according to the layering of taxonomy trees. Let H be the maximum height among taxonomy trees. To facilitate the estimation, other taxonomy trees with height less than H are modified by making their height H.

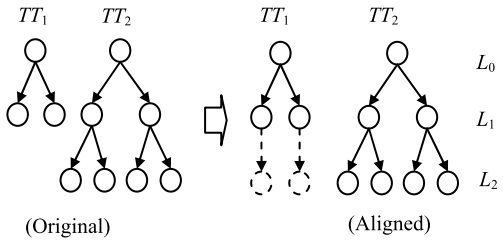


Fig. 2. An example of aligning the height of taxonomy trees.

A string of dummy nodes are attached to their leaf nodes, in order to align them to the maximum height. An example is demonstrated in Fig. 2 to illustrate the above process. The tree TT_2 has the highest tree height 2. As TT_1 has height less than 2, two dummy nodes (dashed circles in the right part of Fig. 2) are added after modification. The height of aligned version of TT_1 is also 2.

The data records in a data set D are logically partitioned by the domain value nodes in taxonomy trees. Let L_j denote the jth layer of the taxonomy forest (the set of aligned taxonomy trees). The number of domain values of TT_i on L_j is denoted as N_{ij}. The data set can be partitioned into $\prod_{i=1}^{m} N_{ij}$ QI-groups in regard to the domain values on level L_j. For instance, the data set is partitioned into 4 groups ($2 \times 2 = 4$) on L_1 shown in Fig. 2, and 8 groups ($2 \times 4 = 8$) on L_2. Based on the assumption data sets are evenly distributed, we define the average anonymity of a data set with respect to a layer of the taxonomy forest as follows.

Definition 5 *(Average anonymity).* The average QI-group size of an anonymous data set D^* with respect to a layer L_j of the taxonomy forest is defined as average anonymity, denoted as K_j, and $K_j = |D^*| / \prod_{i=1}^{m} N_{ij}$. In this case, the current anonymization level exactly consists of all the generalizations/specializations on L_j.

To roughly estimate K, we narrow down the average anonymity interval where K is located in. In terms of assumption of data distribution and Definition 5, it is observed that the anonymization level of any anonymous data set consists generalizations/specializations on at most TWO continuous layers, i.e., the anonymity of the data set must lie between two continuous average anonymity. This observation is quite helpful to estimate K.

To estimate the point where the amount of computation required by MRTDS and MRBUG are equal, we should calculate the computations of them firstly when an anonymity parameter is given. The computations required by MRTDS and MRBUG to achieve K_j-anonymous on level L_j are estimated as follows, respectively. The performance of each round in both MRTDS and MRBUG is mainly dominated by the anonymity computation of the data set because the *Map* functions in both algorithms emit $O(m)$ transformed records in each round of iteration for the anonymity computation, where m is the number of attributes, while only 1 record or so is emitted for information gain/loss computation. As a result, the computations required in one round of MRTDS and MRBUG are roughly equal.

Let C_{Unit} denote the amount of computation incurred in one round of MRTDS or MRBUG. In MRTDS, performing a specialization operation will launch IGPL update and incur C_{Unit} units of computation. So, the total amount of computation to specialize D to L_j, denoted as C_j^{TDS} can be estimated by

$$C_j^{TDS} = \sum_{l=1}^{j} \sum_{i=1}^{m} C_{Unit} \cdot N_{il}. \tag{4}$$

In MRBUG, unlike MRTDS, performing a generalization operation triggers ILPG update just with probability rather than definitely. In general, several operations together launch ILPG update and incur C_{Unit} units of computation. On average, we assume $1/\alpha$ operations can do this, where $0 \leqslant \alpha \leqslant 1$. Then, the total amount of computation to generalize D to L_j, denoted as C_j^{BUG} can be estimated by

$$C_j^{BUG} = \sum_{l=j}^{H-1} \sum_{i=1}^{m} C_{Unit} \cdot \alpha \cdot N_{il}. \tag{5}$$

Keep in mind that the anonymity point where $C_j^{TDS} = C_j^{BUG}$ is what we thrive to estimate. The value of j that makes C_j^{TDS} equal to C_j^{BUG} must be located in the interval $[J-1, J]$, where J satisfies the following condition:

$$\begin{cases} C_J^{TDS} \geqslant C_J^{BUG}, \\ C_{J-1}^{TDS} \leqslant C_{J-1}^{BUG}, \end{cases} \quad \text{where } 1 \leqslant J \leqslant H. \tag{6}$$

Note that C_{Unit} does not affect the solution of the inequality group in (6) because it can be reduced in both inequalities. We test each value of J to determine it since J only values among a small number of integers, while solving the inequality formula explicitly is complicated according to (4) and (5).

Once J is identified, we can estimate that the workload balancing point K lies within $[K_J, K_{J-1}]$ in terms of the definition of average anonymity. Let β be the average branch factor between layer L_{J-1} and L_J of the taxonomy forest. The value of β can be estimated according to $\beta = N_I/N_E$, where N_I is number of nodes on L_{J-1}, i.e., $N_I = \sum_{i=1}^{m} N_{i(J-1)}$, and N_E is number of edges between L_{J-1} and L_J, i.e., $N_E = \sum_{i=1}^{m} N_{iJ}$. Approximately, K_j varies exponentially with respect to j, where $1 \leqslant j \leqslant H$. Specifically, the relationship between K_j and j can be approximately described by $K_j = |D|/\beta^{mj}$ according to Definition 5, where each taxonomy tree is assumed to be a full β-ary tree. This approximation can facilitate the estimation of K during the interval $[K_J, K_{J-1}]$ without losing too much accuracy. As such, we can estimate the workload balancing point at the middle position between L_{J-1} and L_J by

$$K = K_J + |D| \cdot \left(1/\beta^{m(j-1)} - 1/\beta^{mj}\right). \tag{7}$$

Once the workload balancing point is estimated, the hybrid approach can easily determine which component is chosen, via comparing K with k-anonymity parameter.

5.3. Skewness-aware workload balancing point adjustment

Note that we assume that data sets are evenly distributed in the last section when estimating the workload balancing point K. In most real data sets, however, the data distribution is usually not even but skew, which impacts the valuing of K. Both top–down specialization and bottom–up generalization are single dimensional partitioning techniques for sub-tree anonymization scheme [4]. According to a theorem in [17], the maximum QI-group size of an anonymous data set D^* resulting from either TDS or BUG is $O(|D^*|)$. An extremely skew data set can lead to large QI-group size. If such a data set is anonymized via TDS, a few specializations can achieve the goal. Intuitively, the more skew a data set is, the component MRTDS is more preferred in the hybrid approach, because specializing skew data set will accomplish the final k-anonymous data set much faster than generalizing low level domain values in MRBUG. Therefore, it is necessary and reasonable to adjust K according to the skewness of data distribution.

We leverage variance of the data distribution to adjust the estimation of workload balancing point K. The variance can capture the dispersion of the distribution of quasi-identifier attributes. If the variance is zero, the data distributions of all attributes will be uniform as assumed in Section 5.2. In this case, K does not need adjustment. If the variance is large, it implies that the distributions of attribute values are odd rather than even, i.e., some domain values take much more data records than others. This case can lead to quick violation of the k-anonymity requirement when performing top–down specialization. Thus, high variance prefers MRTDS, and we should decrease the value of K to increase the chance of selecting MRTDS. Moreover, higher variance means lower K so that MRTDS is more preferred in relation to higher variance. Specifically, we employ a parameter to adjust K, this parameter is defined as adjustment factor.

Definition 6 *(Adjustment factor).* A parameter γ, $0 \leqslant \gamma \leqslant 1$, employed to adjust K is defined as adjustment factor. The adjusted workload balancing point is $\gamma \cdot K$.

The valuing of adjustment factor γ depends on the variance of data distribution of attributes. For technical convenience, we leverage the coefficient of variation of a distribution to calculate the adjustment factor. Let X_i be a random variable indicating the number of an attribute value of the attribute $Attr_i$ in a data set D, where $1 \leqslant i \leqslant m$. Assume there are n_i leaf domain values for $Attr_i$, denoted as $v_{i1}, \ldots, v_{in_i}$, respectively. Let M_{ij} represent the number of the occurrence of v_{ij} in D, where $1 \leqslant j \leqslant n_i$. Then, X_i takes values from $\{M_{ij} \mid 1 \leqslant j \leqslant n_i\}$, with the same probability, i.e., $1/n_i$. Based on the distribution of X_i, the expectation of X_i, denoted as μ_i, is calculated by

$$\mu_i = \frac{1}{n_i} \sum_{j=1}^{n_i} M_{ij}. \tag{8}$$

Furthermore, the standard deviation, denoted as σ_i, is derived based on μ_i by

$$\sigma_i = \sqrt{\frac{1}{n_i} \sum_{j=1}^{n_i} (M_{ij} - \mu_i)^2}. \tag{9}$$

With μ_i and σ_i, the coefficient of variation, denoted as CV_i, can be obtained by

$$CV_i = \sigma_i/\mu_i. \tag{10}$$

The coefficient of variation is a normalized measure of dispersion of a probability distribution. The reason why we employ the coefficient of variation rather than the standard deviation to measure the dispersion is that we have to normalize the dispersion of the distribution of each attribute in order to aggregate them together and derive an overall value as an indicator. Accordingly, we utilize the mean of all coefficients of variation to measure their aggregate effect. Based on this, we calculate the adjustment factor by

$$\gamma = 1 - \frac{1}{m \cdot CV_{max}} \sum_{i=1}^{m} CV_i. \tag{11}$$

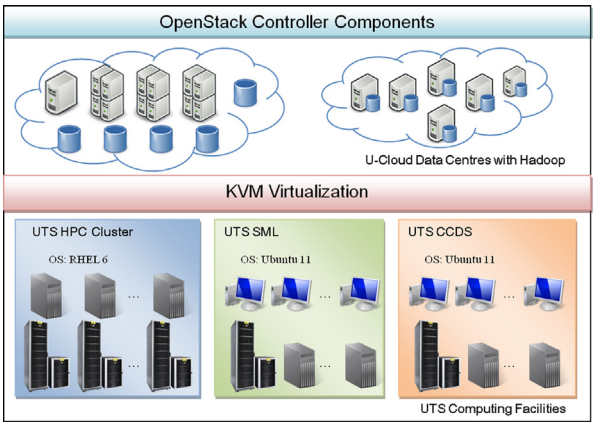


Fig. 3. System overview of U-Cloud.

In (11), CV_{max} is the maximum value among CV_i, $1 \leqslant i \leqslant m$, i.e., $CV_{max} = \max_{1 \leqslant i \leqslant m} CV_i$. Since CV_i can be valued greater than 1, we further normalize them in order to confine their values in $[0, 1]$. Then, the mean of the coefficients of variation of all distributions also ranges in $[0, 1]$. In terms of the aforementioned intuition that γ decreases while the dispersion of the data distributions grows, we define the value of γ to be 1 minus the mean of coefficients of variation. When the variance is 0 (coefficient of variation will be 0 accordingly), γ is valued 1 and no adjustment is required, which is the ideal case where distributions are even as discussed in Section 5.2. To calculate γ, we need to derive M_{ij} from the data set. A simple but MapReduce job that emits each attribute value and counts their occurrence can be developed for this purpose.

6. Experimental evaluation

6.1. Experiment settings

To evaluate the effectiveness and efficiency of the hybrid approach, we compare it with MRTDS [12] and MRBUG. We denote the execution time of the three approaches as T_{Hyb}, T_{TDS} and T_{BUG}, respectively. In general, T_{Hyb} is similar to T_{TDS} if k is larger than K, while T_{Hyb} is similar to T_{BUG} if k is less than K. Estimating K incurs overheads, yet it is minor compared with the computation conducted in MapReduce jobs.

Our experiments are conducted in a cloud environment named U-Cloud. U-Cloud is a cloud computing environment at University of Technology Sydney (UTS). The system overview of U-Cloud has been depicted in Fig. 3. The computing facilities of this system are located among several labs in the Faculty of Engineering and IT, UTS. On top of hardware and Linux operating system (Ubuntu), we install KVM virtualization software [31] which virtualizes the infrastructure and provides unified computing and storage resources. To create virtualized data centers, we install OpenStack open source cloud environment [32] which is responsible for virtual machine management, resource scheduling, task distribution and interaction with users. Furthermore, Hadoop clusters [33] are built based on OpenStack to facilitate MapReduce computing paradigm and big data processing.

All approaches are implemented in Java and standard Hadoop MapReduce API. The Hadoop cluster consists of 20 VMs with type m1.medium which has 2 virtual CPUs and 4 GB Memory. Each round of experiment is repeated 20 times. The mean of the measured results is regarded as the representative. We utilize the Adult data set and its generated data sets like [12].

6.2. Experiment process and results

We measure the change of execution time T_{Hyb}, T_{TDS} and T_{BUG} with respect to anonymity parameter k. The size of data set is set as 1000 MB. Equally, the data set contains 1.1×10^7 data records, which is big enough to evaluate the effectiveness of our approach in terms of data volume or the number of data records. Parameter α is set as 0.5.

Fig. 4 demonstrates the change of T_{Hyb}, T_{TDS} and T_{BUG} with respect to k ranging from 0 to 1.1×10^7. For conciseness, k is indicated by exp which is the exponent of the scientific notation of k, i.e., $k = 1.1 \times 10^{\exp}$. So, exp ranges from 0 to 7. We can see from Fig. 4 that the execution time of the hybrid approach is kept under a certain level, while both MRTDS and MRBUG incur high execution time when k is small and large, respectively. Further, the right part of the curve of T_{Hyb} is near to T_{BUG}, while the right part is near to T_{TDS}. This is because the hybrid approach utilizes MRTDS and MRBUG as components to conduct concrete computation.

As a conclusion, the experimental results demonstrate that the hybrid approach significantly improves the performance of sub-tree anonymization over existing approaches regardless of the k-anonymity parameter.

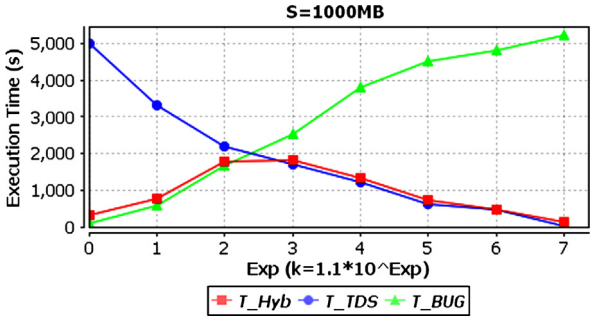


Fig. 4. Change of execution time w.r.t. anonymity parameter k.

7. Conclusion and future work

In this paper, we have investigated the scalability issue of sub-tree anonymization over big data on cloud, and proposed a hybrid approach that combines Top–Down Specialization (TDS) and Bottom–Up Generalization (BUG) together. The hybrid approach automatically selects one of the two components via comparing the user-specified k-anonymity parameter with workload balancing point. Both TDS and BUG have been accomplished in a highly scalable way via a series of deliberately designed MapReduce jobs. Experimental results have demonstrated that the hybrid approach significantly improves the scalability and efficiency of sub-tree data anonymization compared with 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 datasets, thereby requiring intensive investigation. Based on the contributions herein, we plan to further explore the next step on scalable privacy preservation aware analysis and scheduling on large-scale datasets.

References

[1] S. Chaudhuri, What next?: A half-dozen data management research goals for big data and the cloud, in: Proceedings of the 31st Symposium on Principles of Database Systems, PODS'12, 2012, pp. 1–4.
[2] L. Wang, J. Zhan, W. Shi, Y. Liang, In cloud, can scientific communities benefit from the economies of scale?, IEEE Trans. Parallel Distrib. Syst. 23 (2) (2012) 296–303.
[3] H. Takabi, J.B.D. Joshi, G. Ahn, Security and privacy challenges in cloud computing environments, IEEE Secur. Priv. 8 (6) (2010) 24–31.
[4] B.C.M. Fung, K. Wang, R. Chen, P.S. Yu, Privacy-preserving data publishing: A survey of recent developments, ACM Comput. Surv. 42 (4) (2010) 1–53.
[5] B.C.M. Fung, K. Wang, P.S. Yu, Anonymizing classification data for privacy preservation, IEEE Trans. Knowl. Data Eng. 19 (5) (2007) 711–725.
[6] N. Mohammed, B. Fung, P.C.K. Hung, C.K. Lee, Centralized and distributed anonymization for high-dimensional healthcare data, ACM Trans. Knowl. Discov. Data 4 (4) (2010) 18.
[7] B. Fung, K. Wang, L. Wang, P.C.K. Hung, Privacy-preserving data publishing for cluster analysis, Data Knowl. Eng. 68 (6) (2009) 552–575.
[8] W. Ke, P.S. Yu, S. Chakraborty, Bottom–up generalization: A data mining solution to privacy protection, in: Proceedings of 4th IEEE International Conference on Data Mining, ICDM'04, 2004, pp. 249–256.
[9] L. Sweeney, K-anonymity: A model for protecting privacy, Int. J. Uncertain. Fuzziness Knowl.-Based Syst. 10 (5) (2002) 557–570.
[10] J. Dean, S. Ghemawat, MapReduce: A flexible data processing tool, Commun. ACM 53 (1) (2010) 72–77.
[11] Amazon Web Services, Amazon Elastic MapReduce (Amazon EMR), http://aws.amazon.com/elasticmapreduce/, accessed on June 10, 2012.
[12] X. Zhang, L.T. Yang, C. Liu, J. Chen, A scalable two-phase top–down specialization approach for data anonymization using MapReduce on cloud, IEEE Trans. Parallel Distrib. Syst. 25 (2) (2014) 363–373.
[13] A. Machanavajjhala, D. Kifer, J. Gehrke, M. Venkitasubramaniam, L-diversity: Privacy beyond k-anonymity, ACM Trans. Knowl. Discov. Data 1 (1) (2007) 3.
[14] N. Li, T. Li, S. Venkatasubramanian, Closeness: A new privacy measure for data publishing, IEEE Trans. Knowl. Data Eng. 22 (7) (2010) 943–956.
[15] X. Xiao, Y. Tao, M-invariance: Towards privacy preserving re-publication of dynamic datasets, in: Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data, SIGMOD'07, 2007, pp. 689–700.
[16] K. Wang, B.C.M. Fung, P.S. Yu, Handicapping attacker's confidence: An alternative to k-anonymization, Knowl. Inf. Syst. 11 (3) (2007) 345–368.
[17] K. LeFevre, D.J. DeWitt, R. Ramakrishnan, Mondrian multidimensional k-anonymity, in: Proceedings of 22nd International Conference on Data Engineering, ICDE'06, 2006, p. 25.
[18] J. Xu, W. Wang, J. Pei, X. Wang, B. Shi, A.W.-C. Fu, Utility-based anonymization for privacy preservation with less information loss, ACM SIGKDD Explor. Newsl. 8 (2) (2006) 21–30.
[19] X. Xiao, Y. Tao, Anatomy: Simple and effective privacy preservation, in: Proceedings of 32nd International Conference on Very Large Data Bases, VLDB'06, 2006, pp. 139–150.
[20] T. Li, N. Li, J. Zhang, I. Molloy, Slicing: A new approach for privacy preserving data publishing, IEEE Trans. Knowl. Data Eng. 24 (3) (2012) 561–574.
[21] M. Terrovitis, J. Liagouris, N. Mamoulis, S. Skiadopolous, Privacy preservation by disassociation, Proc. VLDB Endow. 5 (10) (2012) 944–955.
[22] K. LeFevre, D.J. DeWitt, R. Ramakrishnan, Incognito: Efficient full-domain k-anonymity, in: Proceedings of 2005 ACM SIGMOD International Conference on Management of Data, SIGMOD'05, 2005, pp. 49–60.
[23] N. Mohammed, B.C.M. Fung, K. Wang, P.C.K. Hung, Privacy-preserving data mashup, in: Proceedings of 12th International Conference on Extending Database Technology: Advances in Database Technology, EDBT'09, 2009, pp. 228–239.
[24] K. LeFevre, D.J. DeWitt, R. Ramakrishnan, Workload-aware anonymization techniques for large-scale datasets, ACM Trans. Database Syst. 33 (3) (2008) 1–47.
[25] T. Iwuchukwu, J.F. Naughton, K-anonymization as spatial indexing: Toward scalable and incremental anonymization, in: Proceedings of the 33rd International Conference on Very Large Data Bases, VLDB'07, 2007, pp. 746–757.
[26] A. Ene, S. Im, B. Moseley, Fast clustering using MapReduce, in: Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD'11, 2011, pp. 681–689.

[27] I. Palit, C.K. Reddy, Scalable and parallel boosting with MapReduce, IEEE Trans. Knowl. Data Eng. 24 (10) (2012) 1904–1916.
[28] R. Vernica, M.J. Carey, C. Li, Efficient parallel set-similarity joins using MapReduce, in: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, SIGMOD'10, 2010, pp. 495–506.
[29] X. Xiao, Y. Tao, Personalized privacy preservation, in: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, SIGMOD'06, 2006, pp. 229–240.
[30] K. Shvachko, K. Hairong, S. Radia, R. Chansler, The Hadoop distributed file system, in: Proceedings of 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies, MSST'10, 2010, pp. 1–10.
[31] KVM, http://www.linux-kvm.org/page/Main_Page, accessed on February 10, 2012.
[32] OpenStack, http://openstack.org/, accessed on February 10, 2012.
[33] Apache, Hadoop, http://hadoop.apache.org, accessed on June 01, 2012.