MR-RBAT: Anonymizing Large Transaction Datasets Using MapReduce

Neelam Memon, Jianhua Shao

MR-RBAT: Anonymizing Large Transaction Datasets Using MapReduce

Neelam Memon and Jianhua Shao
{MemonNG, ShaoJ}@cardiff.ac.uk

School of Computer Science & Informatics
Cardiff University, UK

Abstract. Privacy is a concern when publishing transaction data for applications such as marketing research and biomedical studies. While methods for anonymizing transaction data exist, they are designed to run on a single machine, hence not scalable to large datasets. Recently, MapReduce has emerged as a highly scalable platform for data-intensive applications. In the paper, we consider how MapReduce may be used to provide scalability in transaction anonymization. More specifically, we consider how RBAT may be parallelized using MapReduce. RBAT is a sequential method that has some desirable features for transaction anonymization, but its highly iterative nature makes its parallelization challenging. A direct implementation of RBAT on MapReduce using data partitioning alone can result in significant overhead, which can offset the gains from parallel processing. We propose MR-RBAT that employs two parameters to control parallelization overhead. Our experimental results show that MR-RBAT can scale linearly to large datasets and can retain good data utility.

1 Introduction

Publishing transaction data is important to applications such as personalized web search or understanding purchasing trends. A transaction is a set of items associated with an individual, e.g. search query logs, diagnosis codes or shopping cart items. Such data often contain person-specific sensitive information which may be disclosed in the process. De-identification (i.e. removing personal identifiers) may not provide sufficient protection for individuals' privacy [3, 14], and attack on de-identified data can still lead to two forms of disclosure: *identity disclosure* where an individual can uniquely be linked to their transactions in the dataset and *sensitive item disclosure* where sensitive information about an individual is learnt with or without identifying their transactions.

One approach to protecting privacy is anonymization [10]. Various methods have been proposed [15, 17, 11, 5, 19, 13], and RBAT [13] is one of them that has some desirable features. RBAT is able to deal with both types of disclosure, and can retain high data utility by allowing fine-grained privacy requirements to be specified. However, RBAT is designed to work in a centralized setting and

requires the whole dataset to be memory-resident, on a single machine throughout the anonymization process. This makes RBAT un-scalable to large datasets. For instance, Walmart [18] handles more than one million customers every hour, collecting an estimated 2.5 petabytes of transaction data in the process. RBAT is unable to handle datasets of this scale.

Recently, MapReduce [7] has emerged as a scalable and cost-effective data-processing platform for data-intensive applications. In this paper, we study how MapReduce may be used to improve RBAT's scalability. This is not straightforward due to the highly iterative tasks that RBAT performs. MapReduce does not support iterative processing well [12, 4], and computation in each iteration must be configured separately. This can generate a significant overhead which, if not managed, can offset the gains from parallel processing. To achieve high scalability while ensuring reasonable response time, we propose MR-RBAT, a MapReduce version of RBAT, which implements the key operations of RBAT in parallel and employs two parameters to control the overhead generated by MapReduce. Our experimental results show that MR-RBAT can scale linearly to large datasets and can retain good data utility.

The rest of the paper is organized as follows. Section 2 discusses the related work. In Section 3, we introduce some notations and give an overview of RBAT. Section 4 describes MR-RBAT that we propose in this paper. The experimental results are given in Section 5. Finally, Section 6 concludes the paper.

2 Related Work

Different privacy models have been proposed to guard transaction datasets against disclosure attacks, for example, k^m-anonymity [15, 17], l^m-diversity [16], complete k-anonymity [11], ρ-uncertainty [5], (h, k, p)-Coherence [19] and PS-rules [13]. These models differ in terms of how they assume that the data may be attacked, e.g. k^m-anonymity assumes that an attacker knows up to m items in a transaction and ρ-uncertainty does not distinguish between public and sensitive items, but their sanitization approach is largely similar, relying on some form of data distortion. In this paper, we do not propose yet another privacy model, but instead we focus on the parallelization of the data sanitization method adopted by RBAT.

Recently, there has been a considerable interest in MapReduce as a scalable platform for data intensive applications. One issue that has received much attention is how to handle iterations efficiently in MapReduce. One approach is to extend the standard MapReduce framework itself. Ekanayake et al. [8] and Bu et al. [4], for example, proposed to avoid reading unnecessary data repeatedly from distributed storage, by identifying invariant data and keeping them locally over iterations. However, such methods can only be levered when most of the data remain static between iterations. Iterations of RBAT do not satisfy this requirement. Furthermore, they need to limit some features of the standard MapReduce framework, for example, forcing the data to remain locally means that tasks involving such data cannot be scheduled to be processed on multiple

computing nodes. Such limitations can result in poor performance, especially over heterogeneous clusters.

Other works have proposed to deal with iterative computations in MapReduce algorithmically. Chierichetti et al. [6] implemented an existing greedy Max-k-cover algorithm using MapReduce efficiently and achieved provably approximation to sequential results. Bahmani et al. [2] obtained a parallel implementation of K-means++ [1] and empirically showed to have achieved similar results in a constant number of rounds. MapReduce solutions have also been proposed for anonymization [20, 22, 21], but were limited to achieving k-anonymity for relational data only. Our work is similar to these works in that we also address the iteration issue algorithmically, but the existing methods, including those designed to achieve k-anonymity, cannot trivially be adopted to parallelize the operations performed by RBAT.

3 Background

In this section, we first present some basic notations and concepts necessary to understand our proposed solution, then give a brief overview of RBAT.

3.1 Preliminaries

Let D be a collection of transactions. Each transaction $t \in D$ is a non-empty subset of $I = \{i_1, \cdots, i_n\}$ where each $i_j \in I$, $1 \leq j \leq n$ is called an item. Any $\lambda \subseteq I$ is called an itemset.

Definition 1 (Support). *Given an itemset λ, its support in D, denoted by $\sigma_D(\lambda)$, is the number of transactions containing λ, that is, $\sigma_D(\lambda) = |\{t \in D \ \wedge \ \lambda \subseteq t\}|$.*

We partition I into two disjoint subsets P and S such that $P \cup S = I$ and $P \cap S = \varnothing$. S contains items that are sensitive about the associated individuals and P contains all other items called public items. We assume that S needs to be published intact and that an attacker may have knowledge about individuals in the form of P.

When a set of transactions is released in its original form, certain combinations of public items may not appear frequently enough. This allows an adversary to link an individual to a small set of transactions, thereby breaching privacy. To protect this, PS-rules may be specified [13].

Definition 2 (PS-rule). *Given two itemsets $p \subseteq P$ and $s \subseteq S$, a PS-rule is an implication of the form $p \rightarrow s$.*

Each PS-rule captures an association between a public and a sensitive itemset. The antecedent and consequent of each rule can consist of any public and sensitive items respectively and many PS-rules can be specified by data publishers to capture detailed privacy requirements. A published transaction dataset is deemed to be protected if the specified PS-rules are protected.

Definition 3 (Protection of PS-rule). *Given a dataset D, the parameters $k \in [2, |D|]$ and $c \in [0, 1]$, a PS-rule $p \rightarrow s$ is protected in D if 1) $\sigma_D(p) \geq k$ and 2) $Conf(p \rightarrow s) \leq c$ where $Conf$ is defined as $\frac{\sigma_D(p \cup s)}{\sigma_D(p)}$.*

Condition 1 protects data against identity disclosure by ensuring that the probability of associating an individual to his or her transaction in D using the antecedent of any PS-rule is no more than $1/k$. Condition 2 prevents sensitive item disclosure by ensuring that the probability of linking an individual to a set of sensitive items specified by the consequent of a PS-rule is at most c, given that the probability of associating an individual to his or her transaction using the rule's antecedent is no more than $1/k$.

Given a set of transactions D and a set of PS-rules Θ, if any rule in Θ is not protected, then D must be sanitized. One sanitization approach is set-based generalization which attempts to hide an original item by replacing it with a set of items. It has been shown that set-based generalization retains data utility better than other generalization methods [13]. Consider D given in Table 1, for example. Suppose that we require $k = 3$ and $c = 0.6$. PS-rule $ac \rightarrow h$ is not protected in D as ac has a support of 1 only. But $ac \rightarrow h$ is protected in $\tilde{D}$ given in Table 2 where items a, b and f are replaced by (a, b, f) and c, d and e by (c, d, e) following the generalization, since ac is now supported by 4 transactions and $Conf(ac \rightarrow h) = 0.5$.

Table 1. Original dataset D

Diagnosis Codes
b, c, e, **g**, **h**
a, c, d, **i**, **j**
a, f, **l**
b, e, **g**, **h**
d, f, **l**

Table 2. Anonymized dataset $\tilde{D}$

Diagnosis Codes
(a,b,f), (c,d,e), **g**, **h**
(a,b,f), (c,d,e), **i**, **j**
(a,b,f), **l**
(a,b,f), (c,d,e), **g**, **h**
(c,d,e), (a,b,f), **l**

It is easy to see that there can be many possible generalizations of a dataset to protect a set of PS-rules. The one that incurs least distortion (or has a minimum loss of information) is preferred. RBAT uses the following measure to capture the loss of information as a result of generalization.

Definition 4 (Utility Loss). *Given a generalized dataset $\tilde{D}$, the utility loss of a single generalized item $\tilde{i}$ is given by $UL(\tilde{i}) = \frac{2^{|\tilde{i}|}-1}{2^{|P|}-1} \times w(\tilde{i}) \times \sigma_{\tilde{D}}(\tilde{i})$. The utility loss of the whole dataset $\tilde{D}$ is calculated as $\sum_{\forall \tilde{i} \in \tilde{P}} UL(\tilde{i})$, where $\tilde{P}$ is a set of all generalized items in $\tilde{D}$.*

The UL measure given in Definition 4 captures the loss of information in terms of the size of the generalized itemset, its significance (weight) and its support in $\tilde{D}$. The more items are generalized together, the more uncertain we are about its original representation, hence more utility loss. $w(\tilde{i})$ assigns some penalty based on the importance of the items in $\tilde{i}$. The support of the

generalized item also affects the utility of anonymized data. The more frequently the generalized item occurs in $\tilde{D}$, the more distortion to the whole dataset is.

3.2 The RBAT algorithm

RBAT [13] is a heuristic method for anonymizing transaction data. It is based on the PS-rule privacy model and uses set-based generalization in data sanitization. The key steps of RBAT are given in Algorithm 1.

Algorithm 1 RBAT (D, Θ, $\tilde{i}$, k, c)

Input: Original dataset D, a set of PS-rules Θ, the most generalized item $\tilde{i}$, minimum support k and maximum confidence c.
Output: Anonymized dataset $\tilde{D}$

1: Q.enqueue($\tilde{i}$), $\tilde{D} \leftarrow$ GENERALIZE($D, \tilde{i}$)
2: **while** $|Q| > 0$ **do**
3: $\quad \tilde{i} \leftarrow Q$.dequeue()
4: $\quad \{\tilde{i_l}, \tilde{i_r}\} \leftarrow$ SPLIT($\tilde{i}$)
5: $\quad D' \leftarrow$ UPDATE($\tilde{i_l}$, $\tilde{i_r}$, $\tilde{i}$, $\tilde{D}$, D)
6: $\quad$ **if** CHECK(D', Θ, k, c) = true **then**
7: $\quad\quad Q$.enqueue($\tilde{i_l}$), Q.enqueue($\tilde{i_r}$), $\tilde{D} \leftarrow D'$
8: $\quad$ **end if**
9: **end while**
10: **return** $\tilde{D}$

RBAT is iterative and works in a top-down fashion. Starting with all public items mapped to a single most generalized item $\tilde{i}$ and D generalized to $\tilde{D}$ according to $\tilde{i}$ (step 1), each iteration involves replacing a generalized item $\tilde{i}$ with two less generalized items $\tilde{i_l}$ and $\tilde{i_r}$. RBAT does this greedily by using a two-step split phase (step 4). The first step finds a pair of items from $\tilde{i}$ incurring maximum UL when generalized together. The second step uses the pair as seeds to split $\tilde{i}$ into two disjoint subsets $\tilde{i_l}$ and $\tilde{i_r}$.

To ensure that the anonymized data after replacing $\tilde{i}$ with $\tilde{i_l}$ and $\tilde{i_r}$ still offers the required privacy protection, each split is followed by an update (step 5) and check phase (step 6-8). The update step creates a temporary dataset D' by copying $\tilde{D}$ and replacing $\tilde{i}$ with $\tilde{i_l}$ and $\tilde{i_r}$. D' is then checked to see if Θ is still protected. If it is, D' becomes new $\tilde{D}$ and $\tilde{i_l}$, $\tilde{i_r}$ are queued for further split. The split-update-check is repeated until $|Q| = 0$, in which case $\tilde{D}$ is returned as the result. This top-down specialization process effectively constructs a binary *Split Tree* with the root representing the most generalized item and set of all leaf nodes forming a *Split Cut* representing the final generalization.

4 MR-RBAT

In this section, we describe MR-RBAT. We assume that there are sufficient processing nodes to store all of the data across them and to run all map and reduce

tasks in parallel. Algorithm 2 shows the overall structure of MR-RBAT, which mirrors RBAT (to retain its useful properties), but performs its key computations in parallel (to address scalability). Algorithm 2 is performed on a single processing node as a control, but the functions indicated by an $\mathcal{MR}$ subscript are performed in parallel using MapReduce. In the following sections, we explain the key steps of MR-RBAT.

Algorithm 2 MR-RBAT (D, Θ, $\tilde{i}$, k, c)

Input: Original dataset D, a set of PS-rules Θ, the most generalized item $\tilde{i}$, minimum support k and maximum confidence c.
Output: Anonymized dataset $\tilde{D}$

1: $\tilde{P} \leftarrow \textsc{ComputeUL}_{\mathcal{MR}}(P, D)$
2: $\tilde{D} \leftarrow \textsc{Generalize}_{\mathcal{MR}}(D, \tilde{i})$
3: Q.enqueue($\tilde{i}$)
4: **while** $|Q| > 0$ **do**
5: $\quad \tilde{i} \leftarrow Q$.dequeue()
6: $\quad \{\tilde{i_l}, \tilde{i_r}\} \leftarrow \textsc{Split}_{\mathcal{MR}}(\tilde{i}, \tilde{P}, \tilde{D})$
7: $\quad D' \leftarrow \textsc{Update}_{\mathcal{MR}}(\tilde{i_l}, \tilde{i_r}, \tilde{i}, \tilde{D}, D)$
8: $\quad$ **if** $\textsc{Check}_{\mathcal{MR}}(D', \Theta, k, c) =$ true **then**
9: $\quad\quad Q$.enqueue($\tilde{i_l}$), Q.enqueue($\tilde{i_r}$), $\tilde{D} \leftarrow D'$
10: $\quad$ **end if**
11: **end while**
12: **return** $\tilde{D}$

4.1 Data Partitioning and Preparation

We partition D among M mappers equally using a horizontal file-based data partitioning strategy. That is, first n transactions of D are assigned to the first mapper, the next n to the second mapper, and so on, where $n = \lceil |D|/M \rceil$. This strategy has been shown to be more efficient than other methods [9]. Note that our partitioning method is based on the number of transactions only and does not take into account mappers' memory usage. However, this can be trivially accounted for.

For efficiency, we prepare two datasets, $\tilde{P}$ and $\tilde{D}$, before anonymizing D. We first compute $\tilde{P}$ to contain pairwise ULs of all public items in P (step 1) and then generalize D into $\tilde{D}$ according to the most generalized item $\tilde{i}$. Both are performed using a single MapReduce round and are straightforward, so we will not discuss them further. The benefit of having these computed beforehand will become evident when we discuss the split and update functions later.

4.2 $\textbf{Split}_{\mathcal{MR}}$

This corresponds to the split phase of RBAT (step 4 of Algorithm 1) and is carried out in two steps. The first step uses a single MapReduce round with M

mappers and a single reducer to find a pair which when generalized together incurs maximum UL (Algorithm 3). Each mapper reads a subset of $\mathcal{P}$ from the distributed file system (DFS), finds the pair with maximum UL locally, and sends it to a single reducer (steps 2-3) which finds the pair $\langle i_x, i_y \rangle$ with maximum UL globally (step 5).

Algorithm 3 FINDMAXPAIR ($\tilde{i}$)

Input: A generalized item $\tilde{i}$ to split.
Output: A pair $i_x, i_y \in \tilde{i}$ such that $UL(i_x, i_y)$ is maximum.

1: **Map**$(m, \mathcal{P}_m)$
2: $\quad \mathcal{P}_m \leftarrow$ Load the m-th $\tilde{P}$ from DFS
3: $\quad$ EMIT$(\varnothing, \langle i_a, i_b \rangle)$ such that $UL(i_a, i_b)$ is maximum in $\mathcal{P}_m$
4: **Reduce**$(\varnothing, [\langle i_{a_1}, i_{b_1} \rangle, \langle i_{a_2}, i_{b_2} \rangle, \ldots, \langle i_{a_M}, i_{b_M} \rangle])$
5: $\quad \langle i_x, i_y \rangle \leftarrow \langle i_{a_j}, i_{b_j} \rangle$ such that $UL(i_{a_j}, i_{b_j})$ is maximum, $1 \leq j \leq M$
6: $\quad$ EMIT$(\varnothing, \langle i_x, i_y \rangle)$

The second step uses $\langle i_x, i_y \rangle$ to split $\tilde{i}$ into two less generalized items $\tilde{i_l}$ and $\tilde{i_r}$. RBAT does this by assigning i_x and i_y to I_l and I_r first, then considering each item $i_q \in \tilde{i} - \{i_x, i_y\}$[1] in turn and assigning it to either I_l or I_r based on $UL(I_l \cup i_q)$ and $UL(I_r \cup i_q)$. A direct parallelization of this heuristic will require $|\tilde{i}| - 2$ MapReduce rounds, as the assignment of each item is recursively dependent on the assignment of the items preceding it. In the worse case when the most generalized item is split to single items, one per iteration, it will require a total of $O(|P|^2)$ MapReduce rounds. This will result in a significant setup and data loading overhead.

Alternatively, one may split $\tilde{i}$ based on seeds only. That is, we decide whether an item $i_q \in \tilde{i}$ should be assigned to I_l or I_r based on $UL(i_q, i_x)$ and $UL(i_q, i_y)$. This would then require only a single MapReduce round to split $\tilde{i}$. While this can cut the number of MapReduce rounds significantly, it may cause substantial data utility loss. Consider an extreme case where $\tilde{i} = \{i_1, \ldots, i_{|P|}\}$ is the most generalized item, i_1 and $i_{|P|}$ are the seeds, $\sigma_D(i_j) < k/4, j < |P|$ and $k/2 < \sigma_D(i_{|P|}) < k$. Assuming that a uniform weight of 1 is used in UL calculation, then it is easy to see that using this strategy all the items will be generalized with i_1, resulting in $\tilde{i_l} = (i_1, \cdots, i_{(|P|-1)})$ and $\tilde{i_r} = (i_{|P|})$. As $\sigma_D(i_{|P|}) < k$, $\tilde{i}$ cannot be split, and the data has to be generalized using the most generalized item $\tilde{i}$, incurring a substantial utility loss.

Splitting $\tilde{i}$ by seeds or by preceding items in fact represent two *extreme* strategies for parallelizing split: one has the potential to retain utility better and the other incurs least parallelization overhead. We propose a control that allows a specified number of MapReduce rounds to be run, thereby balancing efficiency and utility retention.

[1] In the rest of this paper, when the context is clear, we shall simply use $\tilde{i}$ instead of $\tilde{i} - \{i_x, i_y\}$ to refer to the set of items in $\tilde{i}$ to be split.

Definition 5 (α-Split). *Given a generalized item $\tilde{i}$ and a pair of seeds i_x and i_y, α-split, $1 \leq \alpha \leq |\tilde{i}|-2$, partitions $\tilde{i}$ into α buckets and splits $\tilde{i}$ in α iterations. Items in each bucket are split based on seeds only, and the splits obtained from the previous iterations are used as the seeds in the current iteration.*

Algorithm 4 shows how α-Split works. $\tilde{i}$ is partitioned into α disjoint buckets (step 2), and α MapReduce rounds are used to split $\tilde{i}$ (step 3). Within each round, each mapper reads a copy of I_l, I_r, bucket $\tilde{i}_h$ and a subset of D, computes the partial support of $I_l \cup i_q$ and $I_r \cup i_q$ for each item $i_q \in \tilde{i}_h$ locally, and then shuffles the results to the reducers (steps 4-9). Each reducer aggregates the partial supports for i_q, assigns i_q to I_l or I_r based on their UL values, and emits updated I_l and I_r as seeds for the next iteration (steps 11-16). Note that currently $\tilde{i}$ is partitioned randomly, i.e. the first $\frac{|\tilde{i}|-2}{\alpha}$ items form the first bucket, the next $\frac{|\tilde{i}|-2}{\alpha}$ items form the second bucket, and so on. Exploring how to best assign items to buckets is beyond the scope of this paper.

Algorithm 4 α-Split ($\tilde{i}$, i_x, i_y, α)

Input: Two seeds i_x, i_y, a generalized item $\tilde{i}$ to split, and split threshold α.
Output: Less generalized items $\tilde{i_l}$ and $\tilde{i_r}$

1: $I_l \leftarrow i_x, I_r \leftarrow i_y$
2: $\{\tilde{i}_1, \tilde{i}_2, \ldots, \tilde{i}_\alpha\} \leftarrow$ PARTITION$(\tilde{i} - \{i_x, i_y\})$
3: **for** $h = 1$ to α **do**
4: **Map**$(m_i, \langle I_l, I_r, \tilde{i}_h, D_m \rangle)$
5: $\mathcal{D}_m \leftarrow$ Load the m-th partition of D from DFS
6: **for** each $i_q \in \tilde{i}_h$ **do**
7: $\tilde{i_l} \leftarrow I_l \cup \{i_q\}, \tilde{i_r} \leftarrow I_r \cup \{i_q\}$
8: EMIT$(i_q, \langle \sigma_{D_m}(\tilde{i_l}), \sigma_{D_m}(\tilde{i_r}) \rangle)$
9: **end for**
10: **Reduce**$(i_q, [\langle \sigma_{D_1}(\tilde{i_l}), \sigma_{D_1}(\tilde{i_r}) \rangle, \langle \sigma_{D_2}(\tilde{i_l}), \sigma_{D_2}(\tilde{i_r}) \rangle, \ldots, \langle \sigma_{D_M}(\tilde{i_l}), \sigma_{D_M}(\tilde{i_r}) \rangle])$
11: **if** $UL(\tilde{i_l}) > UL(\tilde{i_r})$ **then**
12: $I_l \leftarrow I_l \cup \{i_q\}$
13: **else**
14: $I_r \leftarrow I_r \cup \{i_q\}$
15: **end if**
16: EMIT$(r, \langle I_l, I_r \rangle)$
17: **end for**
18: **return** $\langle \tilde{i_l} = I_l, \tilde{i_r} = I_r \rangle$

It is easy to see that α-Split is a generalization of RBAT Split: when $\alpha = |\tilde{i}|$, α-Split becomes RBAT Split. Any other settings of α represent a tradeoff between efficiency and potential utility loss. This gives us a control to balance between performance and quality in anonymizing large transaction datasets, as we will show in Section 5.

We now analyse the overhead cost associated with SPLIT$_{\mathcal{MR}}$. Let $\tilde{i}$ be the item to be split, $s_m(M)$ and $s_r(R)$ be the cost of setting up M mappers and R

reducers, ω be the average time that it takes to read a transaction (of average size in D) from the distributed file system. The overall map cost t_M of a single MapReduce round is given by

$$t_M = s_m(M) + \frac{|D|}{M} \cdot \omega \tag{1}$$

Assume that each mapper has enough parallel connections to send data across the network to R reducers in parallel, the shuffle cost t_S of a single MapReduce round is as follows, where ξ is a network efficiency constant.

$$t_S = \frac{|\tilde{i}| - 2}{\alpha \times min(R, \frac{|\tilde{i}|-2}{\alpha})} \cdot \xi \tag{2}$$

Note that map output with the same key must be sent to the same reducer, so the number of reducers needed is determined by $min(R, \frac{|\tilde{i}|-2}{\alpha})$ in (2). The reduce cost t_R of a single MapReduce round is dominated by the cost of setting up the reducers and reading the shuffled data sent by the mappers:

$$t_R = s_r(R) + \frac{M \times (|\tilde{i}| - 2)}{\alpha \cdot min(R, \frac{|\tilde{i}|-2}{\alpha})} \cdot \omega \tag{3}$$

The overall cost of $\textsc{Split}_{\mathcal{MR}}$ using α iterations to split $\tilde{i}$ is therefore

$$T_{total} = \alpha \times \left[s_m(M) + s_r(R) + \frac{|D|}{M} \cdot \omega + \frac{|\tilde{i}| - 2}{\alpha \times min(R, \frac{|\tilde{i}|-2}{\alpha})} \cdot (\xi + M \cdot \omega) \right] \tag{4}$$

Clearly, a large α, which a direct parallelization of RBAT would imply, can result in a significant overhead cost due to the setup and data loading requirements. We will show in Section 5 that it is possible to use a small α in split to control overhead while retaining good data utility.

4.3 $\textbf{Check}_{\mathcal{MR}}$

Once $\tilde{i}$ is split and $\tilde{D}$ is updated (using a single MapReduce round), Θ must be checked to see if it is still protected. Parallelizing rule-checking while keeping overhead low is another issue to address. RBAT checks all PS-rules in sequence and stops if any rule is found unprotected. Implementing this directly in MapReduce could incur the cost of setting up $O(|\Theta|)$ rounds. We observe that when every rule in Θ is protected, it is more efficient to use a single MapReduce round: mappers check every rule locally and reducers check the overall protection. But when not every rule is protected, this is not efficient. For example, if the first rule in Θ is not protected, then no other rules need to be checked. However, the MapReduce architecture does not allow the nodes to communicate with each other until the whole round is finished, effectively requiring all rules to be checked. This increases the network cost of shuffling partial supports and will incur extra, but unnecessary, computation of checking all the rules.

Again, we observe that checking all rules or one rule only in a single MapReduce round are two extremes of optimisation: checking one rule per round will avoid checking unnecessary rules but can incur severe parallelization overhead, whereas checking all rules in a single round will minimise parallelization overhead but can perform a large amount of unnecessary computation. To balance this, we propose another parameter γ to control the number of MapReduce rounds used to check rules.

Definition 6 (γ-Check). *Given a set of PS-rules Θ, γ-*Check*, $1 \leq \gamma \leq |\Theta|$, checks Θ in γ iterations. Each iteration checks $\frac{|\Theta|}{\gamma}$ PS-rules in Θ.*

Algorithm 5 shows how γ-Check works. It checks $|\Theta|$ in γ MapReduce rounds, each round checking $\frac{|\Theta|}{\gamma}$ rules. Each mapper checks every rule $p \rightarrow s \in \Theta_j$ in a single round, by computing the partial support of $\tilde{p}$ and $\tilde{p} \cup s$ (steps 5-8, where $\phi(p) = \tilde{p}$ generalizes p). The reducers aggregate the partial supports pertaining to each rule and check the protection conditions (steps 9-13). The algorithm will not undergo any subsequent rounds, if any rule is found unprotected in the current round. Hence, γ-Check at maximum checks $(\frac{|\Theta|}{\gamma} - 1)$ more rules than RBAT does, but would require a maximum of γ MapReduce rounds only.

Algorithm 5 γ-Check (D', Θ, k, c, γ)

Input: Temporarily anonymized dataset D', PS-rules Θ, Minimum support k, Maximum confidence c, Check threshold γ.
Output: True if each protected PS-rule and False otherwise.

1: $\{\Theta_1, \Theta_2, \ldots, \Theta_\gamma\} \leftarrow$ PARTITION(Θ)
2: **for** $j \leftarrow 1$ to γ **do**
3: **Map**(m, $\langle D'_m, \Theta_j \rangle$)
4: $\mathcal{D}'_m \leftarrow$ Load the m-th partition of D' from DFS
5: **for** each rule $p \rightarrow s \in \Theta_j$ **do**
6: $\tilde{p} \leftarrow \phi(p)$
7: EMIT $(p \rightarrow s, \langle \sigma_{D'_m}(\tilde{p}), \sigma_{D'_m}(\tilde{p} \cup s) \rangle)$
8: **end for**
9: **Reduce**$(p \rightarrow s, [\langle \sigma_{D'_j}(\tilde{p}), \sigma_{D'_j}(\tilde{p} \cup s) \rangle, 1 \leq j \leq M])$
10: $\sigma_{D'}(\tilde{p}) = \Sigma_{j=1}^{M} \sigma_{D'_j}(\tilde{p}), \quad \sigma_{D'}(\tilde{p} \cup s) = \Sigma_{j=1}^{M} \sigma_{D'_j}(\tilde{p} \cup s)$
11: **if** $\sigma_{D'}(\tilde{p}) < k$ or $\frac{\sigma_{D'}(\tilde{p} \cup s)}{\sigma_{D'}(\tilde{p})} > c$ **then**
12: EMIT(r, False)
13: **end if**
14: **end for**

The cost analysis of CHECK$_{\mathcal{MR}}$ mirrors that of SPLIT$_{\mathcal{MR}}$. That is, Equations (1 - 4) apply to CHECK$_{\mathcal{MR}}$ if we replace α by γ and $|\tilde{i}| - 2$ by Θ, So no further analysis is given here. It is useful to note that γ-Check is a generalization of RBAT Check: when $\gamma = 1$, it becomes RBAT Check. However, unlike the α control, using any γ value in rule checking will not affect the quality of anonymization, but only the performance.

5 Experimental Evaluation

This section describes our experiments. The default settings of parameters are given in Section 5.1 and experimental results are analysed in Section 5.2.

5.1 Setup

All experiments were performed using the Apache Hadoop[2], an open-source implementation of MapReduce. We conducted the experiments over a cloud of thirteen computing nodes, physically located within the same building and interconnected by 100Mbps ethernet connection. One of the machines was allocated to run the MR-RBAT master program (Algorithm 2). All client machines were homogenous in configuration containing 2GB memory and 2 physical cores. Each machine was set to use one core only and was assigned to run a single mapper or reducer instance.

We used a real-world dataset BMS-POS [23] as D, containing several years of point-of-sale transaction records. We have $|D| = 515597$ and $|I| = 1657$ with the maximum and average transaction size being 164 and 6.5, respectively. The larger datasets were constructed by random selection of transactions from D, and we refer to these datasets as nX, where $n \in [0.5, 16]$ is a blow up factor. $|S|$ was set to $0.1 \times |I|$. We used ARE (average relative error) [13] to quantify the utility of anonymized data. We used a workload of 1000 randomly generated queries, each consisting of a pair of public items and a single sensitive item. Default settings of parameters used in the experiments are given in Table 3.

Table 3. Default parameter settings

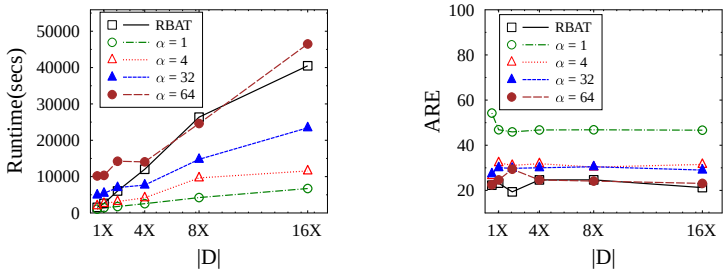
Parameter	Default Value
$\|D\|$	4X
$\|\Theta\|$	4000
k	5
c	0.9
α	4
γ	4

5.2 Scalability and Performance

Figure 1(a) shows how MR-RBAT performed w.r.t. varying data sizes. For smaller datasets, it performed no better than RBAT due to the overwhelming parallelization overhead. However, the run-time of MR-RBAT grows much more slowly than RBAT does, especially when α is small. This demonstrates its scalability to large datasets. It is interesting to observe the run-time for $\alpha = 64$.

[2] http://hadoop.apache.org/

At this setting, MR-RBAT and RBAT achieved almost the same performance. This suggests the severity of overhead caused by iteration using MapReduce: the gains from parallel processing has almost been exhausted entirely by the overhead. It is also interesting to observe ARE results in these experiments. It is expected that setting a smaller α would help performance, but could potentially affect utility. However, Figure 1(b) shows that almost identical utility can be achieved when $\alpha = 64$. This confirms the feasibility and effectiveness of using α in the split phase of MR-RBAT to balance performance and utility retention.

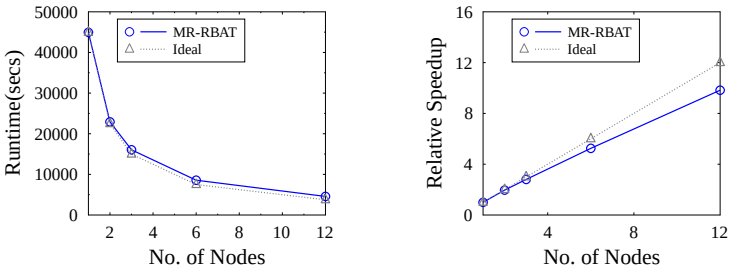


(a) Runtime vs Data Size (b) ARE vs Data Size

Fig. 1. Runtime and ARE of MR-RBAT

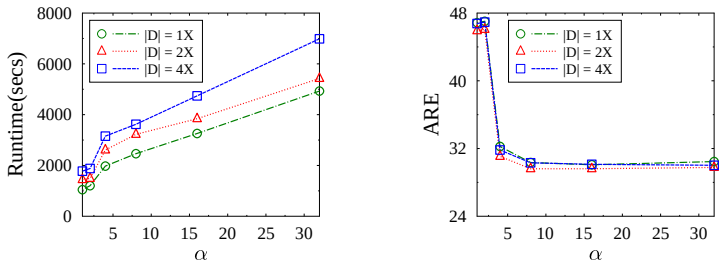
We also evaluated the scalability of MR-RBAT by varying cluster size. Figures 2(a) and 2(b) show the runtime and speedup, respectively. The speedup is measured as the ratio of the runtime of a 1-node cluster to that of the cluster tested. MR-RBAT achieved a near linear speedup when a smaller cluster (relative to data size) was used. This is because MR-RBAT uses one reducer only in the first step of $\textsc{Split}_{\mathcal{MR}}$, and others are used as mappers. Increasing the number of mappers causes more transactions to be fetched and processed by the single reducer, thereby degrading the speedup. Furthermore, we used a dataset of $4X$ in these experiments. With this data size and when cluster size increased, the computation performed by each mapper became lighter, and the effect of overhead on runtime became more significant. So adopting a suitable ratio of data size to cluster size is important to achieving a good speedup.

We then tested the effect of α on runtime and on data utility. As can be seen from Figure 3(a), runtime scales largely linearly w.r.t. α. This suggests that dividing $\tilde{i}$ into different sizes of buckets had little impact on the outcome of split in each iteration. This is further confirmed by the ARE results in Figure 3(b). When α is very small, many items of $\tilde{i}$ are put into one bucket and assigned to I_l or I_r based on seeds only. This caused items with high ULs to be generalized together. But once α is large enough, i.e. buckets are smaller enough, the α value only affected runtime, not ARE. This shows that the performance of splitting $\tilde{i}$ can be significantly improved by using a relatively small α without compromising data utility.



(a) Runtime vs Cluster Size (b) Speedup vs Cluster Size

Fig. 2. Scalability of MR-RBAT



(a) α vs Runtime (b) α vs ARE

Fig. 3. Effect of α

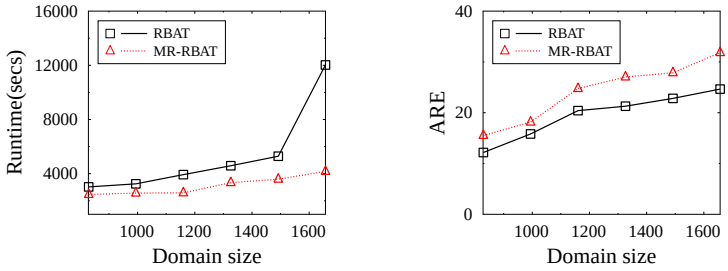
We also observed a relationship between α and split skewness during these experiments. Let S_α be the split tree constructed by MR-RBAT based on some α, and $\tilde{i}$ be a non-leaf node of S_α with $\tilde{i}_l$ and $\tilde{i}_r$ as its left and right children respectively. We measure split skewness $\xi(S_\alpha)$ as

$$\xi(S_\alpha) = \sum_{\tilde{i} \in S_\alpha} || \, |\tilde{i}_l| - |\tilde{i}_r| \, || \tag{5}$$

It was observed that split skewness decreased as α was increased. This is because when α is small, a large bucket of items will be split in a single round based on the seeds only. If data distribution is such that generalizing most of items in the bucket with one of the seeds produces a larger UL than the other seed does, then most of the items will be generalized with one seed, resulting in a skewed split. Skewed splits are more likely to make Θ unprotected, resulting in an early stop in split and a higher ARE. This is confirmed by Figure 3(b). Very small α values are therefore to be avoided.

Next, we varied domain size to see its effect on runtime. As shown in Figures 4(a) and 4(b), MR-RBAT's runtime was more stable and grew much more slowly than RBAT did as the domain size increased. On the other hand, the difference

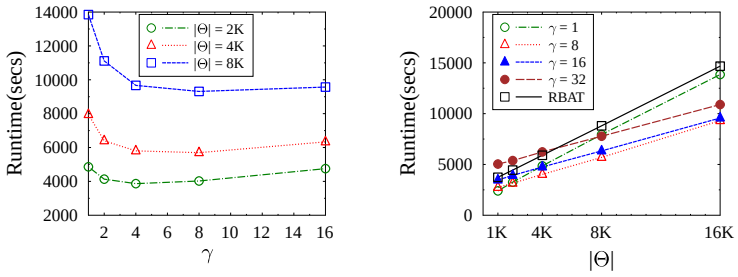
in ARE between RBAT and MR-RBAT increased slightly as the domain size was increased. This is because MR-RBAT uses a fixed number of MapReduce rounds in split. Increasing domain size causes more items to be put in one bucket and generalized in one round. This contributed to an increased ARE.



(a) Domain size vs Runtime (b) Domain size vs ARE

Fig. 4. Effect of $|I|$

Finally, we tested the effect of γ on runtime. Note that a γ setting will only affect runtime, not ARE, so only the runtime results are reported in Figure 5. Observe that initially the runtime was decreased when we increased γ (see Figure 5(a)). This is because when γ is small, the inefficiency introduced by CHECK$_{\mathcal{MR}}$ is mainly from the need to check extra, but unnecessary rules. As γ increases, the number of unnecessary rules to check decreases, resulting in a better runtime. However, as we further increase γ, reduction through checking fewer unnecessary rules decreases, but the cost of setting up more iterations increases, making an increase in the overall runtime.



(a) γ vs Runtime (b) $|\Theta|$ vs Runtime

Fig. 5. Effect of γ

Increasing the number of rules also caused the runtime of MR-RBAT to increase, but much more slowly than RBAT did, except for $\gamma = 1$, as shown in Figure 5(b). When $\gamma = 1$, MR-RBAT checks every rule in Θ and is not efficient for the reason we gave above. All other settings of γ have resulted better runtime and scalability w.r.t the number of rules to be enforced in anonymization.

6 Conclusions

In this paper, we have studied how RBAT may be made scalable using MapReduce. This is important to a range of transaction data anonymization solutions as most of them, like RBAT, are designed for a centralized setting and involve some iterative data distortion operations in data sanitization. We have shown that parallelizing RBAT in MapReduce by some straightforward data partitioning can incur an overwhelmingly high parallelization cost due to the iterative operations to be performed. We proposed MR-RBAT which employ two controls to limit the maximum number of MapReduce rounds to be used during data generalization, thereby reducing the overhead and computational cost. We have empirically studied the effect of different settings of these controls and have found that MR-RBAT can scale nearly linear w.r.t. the size of data and can efficiently anonymize datasets consisting of millions of transactions, while retaining good data utility.

References

1. Arthur, D., and Vassilvitskii, S. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms* (2007), Society for Industrial and Applied Mathematics, pp. 1027–1035.
2. Bahmani, B., Moseley, B., Vattani, A., Kumar, R., and Vassilvitskii, S. Scalable k-means++. *Proceedings of the VLDB Endowment 5*, 7 (2012), 622–633.
3. Barbaro, M., Zeller, T., and Hansell, S. A face is exposed for aol searcher no. 4417749. *New York Times 9*, 2008 (2006), 8For.
4. Bu, Y., Howe, B., Balazinska, M., and Ernst, M. D. Haloop: Efficient iterative data processing on large clusters. *Proc. VLDB Endow. 3*, 1-2 (2010), 285–296.
5. Cao, J., Karras, P., Raïssi, C., and Tan, K.-L. ρ-uncertainty: inference-proof transaction anonymization. *Proceedings of the VLDB Endowment 3*, 1-2 (2010), 1033–1044.
6. Chierichetti, F., Kumar, R., and Tomkins, A. Max-cover in map-reduce. In *Proceedings of the 19th International Conference on World Wide Web* (New York, NY, USA, 2010), WWW '10, ACM, pp. 231–240.
7. Dean, J., and Ghemawat, S. Mapreduce: Simplified data processing on large clusters. *Commun. ACM 51*, 1 (Jan. 2008), 107–113.
8. Ekanayake, J., Li, H., Zhang, B., Gunarathne, T., Bae, S.-H., Qiu, J., and Fox, G. Twister: a runtime for iterative mapreduce. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing* (2010), ACM, pp. 810–818.

9. Ferreira Cordeiro, R. L., Traina, Junior, C., Machado Traina, A. J., López, J., Kang, U., and Faloutsos, C. Clustering very large multi-dimensional datasets with mapreduce. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2011), KDD '11, ACM, pp. 690–698.
10. Fung, B. C. M., Wang, K., Chen, R., and Yu, P. S. Privacy-preserving data publishing: A survey of recent developments. *ACM Comput. Surv. 42*, 4 (June 2010), 14:1–14:53.
11. He, Y., and Naughton, J. F. Anonymization of set-valued data via top-down, local generalization. *Proc. VLDB Endow. 2*, 1 (Aug. 2009), 934–945.
12. Lee, K.-H., Lee, Y.-J., Choi, H., Chung, Y. D., and Moon, B. Parallel data processing with mapreduce: A survey. *SIGMOD Rec. 40*, 4 (Jan. 2012), 11–20.
13. Loukides, G., Gkoulalas-Divanis, A., and Shao, J. Anonymizing transaction data to eliminate sensitive inferences. In *Proceedings of the 21st International Conference on Database and Expert Systems Applications: Part I* (Berlin, Heidelberg, 2010), DEXA'10, Springer-Verlag, pp. 400–415.
14. Narayanan, A., and Shmatikov, V. How to break anonymity of the netflix prize dataset. *CoRR abs/cs/0610105* (2006).
15. Terrovitis, M., Mamoulis, N., and Kalnis, P. Privacy-preserving anonymization of set-valued data. *Proc. VLDB Endow. 1*, 1 (Aug. 2008), 115–125.
16. Terrovitis, M., Mamoulis, N., and Kalnis, P. Local and global recoding methods for anonymizing set-valued data. *The VLDB Journal 20*, 1 (Feb. 2011), 83–106.
17. Terrovitis, M., Mamoulis, N., Liagouris, J., and Skiadopoulos, S. Privacy preservation by disassociation. *Proc. VLDB Endow. 5*, 10 (June 2012), 944–955.
18. The Economist. A special report on managing information: Data, data everywhere. *The Economist* (February 2010).
19. Xu, Y., Wang, K., Fu, A. W.-C., and Yu, P. S. Anonymizing transaction databases for publication. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2008), KDD '08, ACM, pp. 767–775.
20. Zhang, X., Liu, C., Nepal, S., Yang, C., Dou, W., and Chen, J. Combining top-down and bottom-up: Scalable sub-tree anonymization over big data using mapreduce on cloud. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on* (July 2013), pp. 501–508.
21. Zhang, X., Liu, C., Nepal, S., Yang, C., Dou, W., and Chen, J. A hybrid approach for scalable sub-tree anonymization over big data using mapreduce on cloud. *Journal of Computer and System Sciences 80*, 5 (2014), 1008 – 1020. Special Issue on Dependable and Secure Computing The 9th {IEEE} International Conference on Dependable, Autonomic and Secure Computing.
22. Zhang, X., Yang, L., Liu, C., and Chen, J. A scalable two-phase top-down specialization approach for data anonymization using mapreduce on cloud. *Parallel and Distributed Systems, IEEE Transactions on 25*, 2 (Feb 2014), 363–373.
23. Zheng, Z., Kohavi, R., and Mason, L. Real world performance of association rule algorithms. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2001), KDD '01, ACM, pp. 401–406.