A Parallel Method for Scalable Anonymization of Transaction Data

Neelam Memon, Grigorios Loukides, Jianhua Shao
School of Computer Science & Informatics
Cardiff University, UK
{MemonNG, G.Loukides, ShaoJ}@cardiff.ac.uk

***Abstract*—Transaction data, such as market basket or diagnostic data, contain sensitive information about individuals. Such data are often disseminated widely to support analytic studies. This raises privacy concerns, as the confidentiality of individuals must be protected. Anonymization is an established methodology to protect transaction data, which can be applied using different algorithms. RBAT is an algorithm for anonymzitng transaction data that has many desirable features. These include flexible specification of privacy requirements and the ability to preserve data utility well. However, like most anonymization methods, RBAT is a sequential algorithm that is not scalable to large datasets. This limits the applicability of RBAT in practice. To address this issue, in this paper, we develop a parallel version of RBAT using MapReduce. We partition the data across a cluster of computing nodes and implement the key operations of RBAT in parallel. Our experimental results show that scalable anonymization of large transaction datasets can be achieved using MapReduce and our method can scale nearly linear to the number of processing nodes.**

I. Introduction

A transaction dataset is a collection of records, each consisting of a set of items drawn from some domain. Each such record is called a transaction and is assumed to be associated with a single individual. Examples of transactions are search query logs, diagnosis codes, and shopping cart items. Transaction data are often analyzed to support, for example, personalized web search, personalized medicine, or purchasing trend projection. However, as such data often contain sensitive information about individuals, releasing them in their original form may lead to privacy breaches. Thus, some data sanitization must be exercised before the release of transaction data.

There are two forms of privacy disclosure that must be considered when publishing transactions: identity disclosure and sensitive item disclosure. *Identity disclosure* happens when an individual is linked to their de-identified transaction based on some background knowledge. *Sensitive item disclosure* occurs when some sensitive items associated with an individual are learnt with or without identifying their transactions. Different methods have been proposed to guard transaction data against either form of disclosure [1]–[6], but they all require the whole dataset to be memory-resident and to be anonymized using a single centralized machine. This is not scalable. For instance, Walmart handles more than a million customers every hour, collecting an estimated 2.5 petabytes of data in the process [7]. The existing methods are unable to handle data of this scale.

A promising and cost-effective way to achieve scalability is to perform anonymization in parallel. Recently, MapReduce [8] has emerged as a scalable and cost-effective platform for data-intensive applications. In this paper, we consider how MapReduce may be used to achieve scalable transaction data anonymization. More specifically, we consider the implementation of RBAT [6] using MapReduce. RBAT as a transaction data anonymization method has some desirable features. However, like many other transaction anonymization methods, RBAT is sequential and is not scalable to large datasets. To address this, we have designed and experimented with a parallel version of RBAT where data are partitioned across a cluster of computing nodes and the key operations of RBAT are performed in parallel. Our experimental results show that scalable transaction anonymization can be achieved using MapReduce and our method can scale nearly linear to the number of processing nodes.

The rest of the paper is organized as follows. Section II discusses the related work. A brief description of RBAT and Map-Reduce is given in Section III. Section IV presents the parallel version of RBAT. Experimental results are presented in Section V. Section VI concludes the paper.

II. Related Work

Data anonymization is a popular approach to protect data of different forms, including relational and transaction data. There have been different approaches to data anonymization, and we refer the reader to [9]–[11] for surveys. Different privacy models have been proposed to protect transaction datasets, for example, k^m-anonymity [1], l^m-diversity [12], complete k-anonymity [3], ρ-uncertainty [4], (h, k, p)-Coherence [5] and PS-rules [6]. These models differ in their assumptions about how data may be attacked by an adversary, but all can be enforced through data transformation. In this paper, we focus on the scalability of data sanitization, rather than proposing a new privacy model to deal with a specific privacy threat. More specifically, we focus on the PS-rules privacy model, which is employed by the RBAT algorithm.

Scalable data anonymization has been considered in prior work. Iwuchukwu et al. [13] proposed bulk-loading techniques which use an R+-tree index to enhance anonymization performance. Lefevre et al. [14] and Loukides et al. [15]

proposed sampling based methods to anonymize large datasets. These approaches are however designed to work on a single machine, and thus their scalability is limited. In contrast, we propose a parallel solution to address scalability in transaction anonymization.

Recently, there has been a considerable interest in designing MapReduce-based solutions for a range of data-intensive applications, including performing join operations [16]–[19], clustering [20]–[23], and mining association rules [24]. Privacy protection when using MapReduce over clouds has also been studied. Roy et al. [25] proposed Airavat, a system that ensures mandatory access control and differential privacy [26] when performing MapReduce operations over sensitive data. Zhang et al. [27] presented hybrid cloud techniques to support privacy-aware data-intensive MapReduce computations. Zhang et al. [28] incorporated encryption with anonymization to achieve privacy preservation over multiple datasets. Our work is different from these in that they address the issue of privacy when using computing clusters, whereas we focus on the scalability of the anonymization process itself. Closely related to our work are the recent studies by Zhang et al. [29], [30]. They designed MapReduce-based bottom-up [29] and top-down [30] approaches to achieving k-anonymity [31] for relational data, but their methods cannot trivially be adopted to leverage the privacy-utility trade-off that transaction anonymization exercises and demands.

III. RBAT and MapReduce

In this section we give a brief introduction to RBAT and MapReduce that are necessary to understand this paper. The reader is referred to [6] for details of RBAT and to [32] for details of MapReduce.

A. RBAT Algorithm

RBAT is a heuristic method and anonymizes transaction data using set-based generalization [33]. It protects data based on a set of user-specified PS-rules. A PS-rule is an implication of the form $p \rightarrow s$, where p contains public items (assumed to be available to attackers) and s sensitive items. We will refer to p as the antecedent of the rule and to s as the consequent. To ensure that a set of transactions is protected, RBAT requires that p is supported by at least k transactions (i.e. there are at least k transactions containing p) and that the confidence of $p \rightarrow s$ is no more than c (i.e. if p is supported by m transactions and $p \cup s$ by n transactions, then $n/m \leq c$). If this is not the case, then the data will be generalized. For example, item a in one transaction and item b in another may both be replaced by a set (a, b) (called a *generalized item*) to increase support for both a and b.

Given a set of transactions D and a set of PS-rules Θ to be protected, RBAT generalizes D iteratively 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 $\tilde{i}$, RBAT iteratively performs the following three key steps:

- Step 1 (Split): Splitting $\tilde{i}$ into two less generalized itemsets $\tilde{i}_l$ and $\tilde{i}_r$.
- Step 2 (Update): Temporarily generalizing D into D' according to $\tilde{i}_l$ and $\tilde{i}_r$.
- Step 3 (Check): Checking if the set of PS-rules Θ is protected in D'.

If Θ is protected, then D' becomes the new $\tilde{D}$ and the split-update-check process is repeated, but recursively performed on $\tilde{i}_l$ and $\tilde{i}_r$. Otherwise the split process stops. This top-down specialization process effectively creates a binary *Split Tree* with root representing the most generalized item and the set of leaf nodes form the final generalization. It is easy to see that there are many splits of $\tilde{i}$ possible, therefore there are many generalizations of D possible. The generalization that incurs minimum information loss (or least distortion to the data) is preferred. RBAT achieves this heuristically by minimizing an objective measure, called Utility Loss (UL), at each iteration:

$$UL(\tilde{i}) = \frac{2^{|\tilde{i}|} - 1}{2^{|P|} - 1} \times w(\tilde{i}) \times \sigma_{\tilde{D}}(\tilde{i}) \quad (1)$$

$$UL(\tilde{D}) = \sum_{\forall \tilde{i} \in \tilde{P}} UL(\tilde{i}) \quad (2)$$

where $\tilde{P}$ is the set of leaf nodes obtained from the split tree (i.e. the set of generalized items that will be used to generalize D into $\tilde{D}$), $\sigma_{\tilde{D}}(\tilde{i})$ is the support for the generalized item $\tilde{i}$ in $\tilde{D}$, and $w(\tilde{i})$ is the weight representing the importance of the items in $\tilde{i}$.

The UL measure captures the information loss in terms of the size of the generalized item, its significance 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 (i.e., the number of times the generalized item occurs in the dataset) also affects the utility of anonymized data. The more frequently the generalized item occurs in $\tilde{D}$, the more distortion the generalized item incurs to the dataset.

We now illustrate how RBAT works through an example. Consider the anonymization of transaction dataset D given in Table I with $k = 3$, $c = 0.6$ and $\Theta = \{be \rightarrow gh, f \rightarrow l\}$. RBAT starts with the most generalized item $\tilde{i} = (a, b, c, d, e, f)$, i.e. all the public items in D are mapped to this single generalized item.

TABLE I
AN EXAMPLE DATASET D

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

RBAT then attempts to find two less generalized items from $\tilde{i}$. In this case, the pair $\langle a, e \rangle$ is found to incur maximum UL when generalized together, so a and e are used as seeds to split $\tilde{i}$ heuristically into two disjoint subsets $\tilde{i}_l$ and $\tilde{i}_r$. Following

a few iterations and based on UL values, RBAT returns $\tilde{i}_l = (a, b, f)$ and $\tilde{i}_r = (c, d, e)$.

$\tilde{D}$ is now updated with $\tilde{i}$ being replaced by $\tilde{i}_l$ and $\tilde{i}_r$, resulting in a temporal dataset D' as shown in Table II. This dataset is then checked to see if Θ is still protected, that is, if the generalized antecedents of the two rules in Θ (b, e which are generalized to $(a, b, f)(c, d, e)$ and f which is genearlized to (a, b, f)) are still supported by at least $k = 3$ transactions in D' and the confidence of the two rules ($b, e \rightarrow g, h$ and $f \rightarrow l$) are still below $c = 0.6$.

TABLE II
THE DATASET $\tilde{D}$ AFTER ANONYMIZING 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 check that Θ is protected in Table II, so D' becomes the new $\tilde{D}$ and $\tilde{i}_l = (a, b, f)$ and $\tilde{i}_r = (c, d, e)$ are put on the queue for further split. In the next iteration, the same split process is applied to $\tilde{i}_l = (a, b, f)$, resulting in (a, f) and (b). This time, we find that support for generalized b, e is less than $k = 3$, so the split is discarded. $\tilde{i}_r = (c, d, e)$ is then considered in the following iteration and it cannot be split either due to the protection requirement. So RBAT returns Table II as the final $\tilde{D}$.

B. MapReduce

MapReduce [8], [32] achieves parallel computation by configuring a set of shared-nothing mappers and reducers. A schematic representation of a typical MapReduce round is shown in Figure 1.

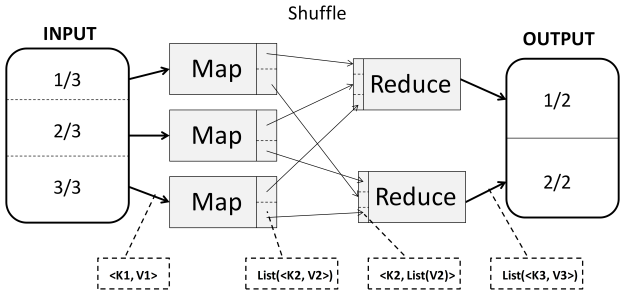

Fig. 1. A schematic representation of a MapReduce round

During the Map stage, the data is partitioned into a number of disjoint subsets and each subset is assigned to a separate mapper. Each mapper (in parallel to other mappers) reads the data, performs the user-specified computation on the subset assigned to it, and outputs intermediate results as a set of $\langle Key, Value \rangle$ pairs. For example, in a simple word count problem, a mapper reads its assigned chunk of text and outputs a $\langle word, occurrences \rangle$ pair for each distinct word.

The intermediate output from the mappers goes through a shuffle phase. The same keys from different mappers are grouped together and are assigned to the same reducer by the shuffle phase. Note that a mapper output may not be local to its assigned reducer. In this case, the shuffle phase also involves transferring mapper output to a reducer over the network. The shuffle phase starts as soon as the first mapper finishes its processing, so it may overlap with the map phase. Each reducer processes the mapper output group using a user-specified reduce function. The output produced by the reduce phase can be the final output or can be input to another mapper in the next MapReduce round.

It is worth noting that there is an overhead for achieving parallelization in MapReduce. This mostly consists of the cost of initializing a MapReduce job, I/O cost of reading input and writing output by mappers and reducers during the map and reduce stages, and the cost of shuffling the mapper output over the network to reducers. It is also worth observing that as the intermediate output produced by the mappers is shuffled over the network and then read by the reducers, the less the intermediate output is produced by the map stage, the lower the cost of transferring the data over the network and reading input by the reducers will be.

IV. PARALLEL RBAT

In this section, we describe our MapReduce version of RBAT. Our method is mainly based on the observation that the support computation required by Steps 1-2 and the dataset updating required by Step 3 of RBAT (see Section III) require the whole dataset to be scanned. Our design partitions the data across the available computing nodes and perform these operations in parallel. We assume that we have enough memory to load all of the input data across the computing nodes in use. In the following, we first describe the data partitioning approach and the support computation operation in parallel, followed by the details of our method.

A. Data Partitioning and Representation

Given a set of transactions D to be anonymized and M mappers available, we partition D among M mappers in a way that workload on each mapper is approximately equal. Generally speaking, there are two approaches to partitioning D: item-based or record-based.

With item-based partitioning, a set of transactions is partitioned vertically based on items. Let I be a set of all public items, we divide I into M disjoint subsets $I_j, 1 \leq j \leq M$, and then hash D into M subsets based on I_j. Each mapper m is assigned a set of pairs $\langle i, v \rangle$, where i is an item in I_j and v is a vector of identifiers of transactions that contain i as an item. Item-based partitioning is efficient for support computation. For example, the support of a generalized item (a, b) from partition I_l containing both a and b will only require to access two sets indexed by a and b, and then perform $\sigma_D(ab) = |a \cup b|$. But in the case where $a \in I_j$ and $b \in I_l$ with $j \neq l$, both mappers need to shuffle the relevant vector

over the network. This may dramatically increase the shuffle cost.

Record-based partitioning, on the other hand, partitions data horizontally by records. Given D has $\{t_1, \ldots, t_{|D|}\}$ transactions, it assigns about $n = \frac{|D|}{M}$ to each mapper. That is, $\{t_1, \cdots, t_n\}$ are assigned to the first mapper, $\{t_{(n+1)}, \cdots, t_{2n}\}$ to the second mapper, and so on. Note that for efficiency purposes, we assign transactions based on their arrival order. Using this approach, each mapper will have a disjoint subset $D_j, 1 \leq j \leq M$, where D_j may contain any items of I. This approach will only require a count to be distributed over the network, but the support computation of a generalized item will require the scanning of the whole partition.

We adopt record-based partitioning in our method as network cost could potentially be far more significant than performing a in-memory scan of the data partition. So we assign each partition D_j to a mapper, and each mapper (in parallel to other mappers) reads its assigned subset for processing. To make the support computation efficient, we represent each data partition D_j using the hashing structure employed by item-based partitioning. That is, we represent D_j as the following hash table:

$$\begin{pmatrix} i_1 & : & t_{1,1} & \cdots & t_{1,\beta_1} \\ i_2 & : & t_{2,1} & \cdots & t_{2,\beta_2} \\ \cdots & & \cdots & \cdots & \cdots \\ i_{|I|} & : & t_{|I|,1} & \cdots & t_{|I|,\beta_{|I|}} \end{pmatrix}$$

where $i_1, \ldots, i_{|I|} \in I$ serve as index, and $t_{ij}, 1 \leq i \leq |I|$, $1 \leq i \leq \beta_j$ are identifiers of transactions containing the index item. This approach ensures low overhead for shuffling the data over the network and will also make the support computation efficient.

$$\begin{pmatrix} t_1 & : & i_{1,1} & \cdots & i_{1,\alpha_1} \\ t_2 & : & i_{2,1} & \cdots & i_{2,\alpha_2} \\ \cdots & & \cdots & \cdots & \cdots \\ t_{|D|} & : & i_{|D|,1} & \cdots & i_{|D|,\alpha_{|D|}} \end{pmatrix}$$

B. Parallel Support Computation

We observe that support computation is the most frequent operation in RBAT that requires full data scan. It is performed to compute the UL of generalized items and to check if PS-rules are protected. Given an generalized item $\tilde{i}$, its support in D, denoted by $\sigma_D(\tilde{i})$, is calculated by

$$\sigma_D(\tilde{i}) = |\{t \in D \wedge \tilde{i} \subseteq t\}| \tag{3}$$

We now show how support is computed using MapReduce. Given D and a set of items $\lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_h\}$ whose supports are to be counted. We perform the support computation in a single MapReduce round. D is partitioned into $\{D_1, \ldots, D_M\}$ using record-based partitioning. Each mapper then iterates over λ, computing and emitting local support of each element $\lambda_a \in \lambda$:

$$Map(D_j, \lambda_1, \ldots, \lambda_{|\lambda|}) \rightarrow [\langle \lambda_1, \sigma_{D_j}(\lambda_1) \rangle \ldots \langle \lambda_{|\lambda|}, \sigma_{D_j}(\lambda_{|\lambda|}) \rangle]$$

These partial supports are shuffled over the network to the corresponding reducers. Each reducer (in parallel to other reducers) accumulates the partial supports corresponding to λ_a and computes the global support.

C. Parallel RBAT

A schematic representation of our parallel RBAT is shown in Figure 2. The first phase shown in Figure 2 corresponds to the split phase (Step 1) of RBAT (see Section III) and is done in parallel 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. Each mapper reads a subset of a pre-computed matrix $\mathcal{P}$ containing ULs of all possible pairs of public items P (Step 2). The pair with maximum UL from each mapper is sent to a single reducer (Step 3) which finds the pair $\langle i_x, i_y \rangle$ with maximum UL globally. $\mathcal{P}$ itself is computed, only once in the beginning using a single MapReduce round.

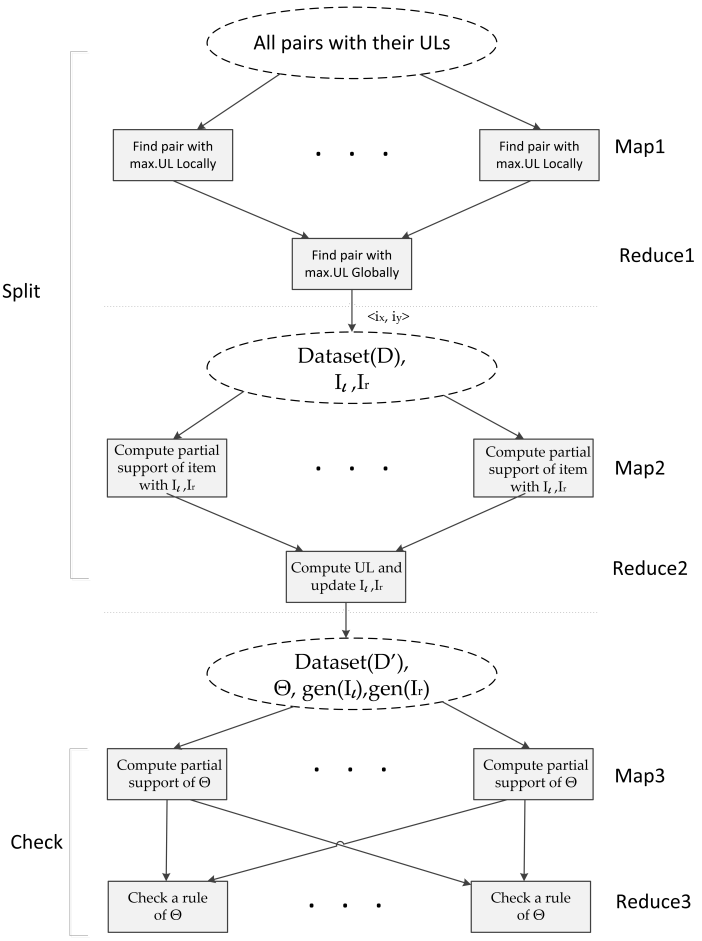

Fig. 2. A schematic representation of Parallel RBAT

The second step is iterative and uses $\langle i_x, i_y \rangle$ to split $\tilde{i}$ into two less generalized items $\tilde{i_l}$ and $\tilde{i_r}$. Initially, I_l and I_r are assigned the seeds i_x and i_y. For every item $i_q \in \tilde{i}$ that is not a seed, a MapReduce round is used to decide whether it should be generalized with I_l or I_r. The M mappers read D_j in parallel and compute $\sigma_{D_j}(I_l \cup \{i_q\})$ and $\sigma_{D_j}(I_r \cup \{i_q\})$. These partial supports from all the mappers are shuffled over the network to a single reducer. The reducer then computes the UL of $I_l \cup \{i_q\}$ and $I_r \cup \{i_q\}$ and adds i_q to either I_l or I_r based on which generalization incurs less UL. Finally, I_l and I_r are returned as two less generalized items.

The updating stage (Step 2) of RBAT (see Section III) is performed in a single Map-only round. Mappers read $\tilde{D}_j$ in parallel and replace every occurrence of $\tilde{i}$ by the current generalizations $\tilde{i_l}$ and $\tilde{i_r}$. Note that we have not shown the update phase in Figure 2 and will not discuss it further as a) it is fairly trivial to perform the updating step in single Map-only round, and b) it does not require access to the whole dataset. Thus, it has little impact on the performance of parallelization of RBAT.

The second phase shown in Figure 2 corresponds to the check phase (Step 3) of RBAT (see Section III). A single MapReduce round is used to check if all PS-rules are protected. Each mapper reads a set of leaves from the split tree constructed so far and a subset of temporarily anonymized transactions D'_j. For each PS-rule $p \rightarrow s$ it generalizes p to $\tilde{p}$, computes the partial support of $\tilde{p}$ and $(\tilde{p} \cup s)$, and shuffles the result over the network to reducers with $p \rightarrow s$ as key. The reducers compute the global support $\sigma_{D'}(\tilde{p})$ and $\sigma_{D'}(\tilde{p} \cup s)$ for each PS-rule, and return true or false based on if the rule is protected or not.

V. Experimental Evaluation

We implemented RBAT and the parallel version of it in C++. The parallel version was implemented using Apache Hadoop[1], an open-source implementation of the MapReduce framework. All the experiments were performed over a cloud consisting 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 master program. All the worker 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 BMS-web-view-1 [15], a real-world click-stream dataset with $|D| = 59602$ and $|I| = 497$. The larger data sizes are acquired by random selection of transactions from the original dataset. 10% of the items were randomly selected as sensitive items. We do not report data utility in this paper, since our parallel version gives exactly the same anonymized results given by RBAT. Unless otherwise specified, our default settings for the experiments are given in Table III

First, we varied the datasize from 8 to 128 million transactions and measured the response time of both RBAT and parallel RBAT. As shown in Figure 3, parallel RBAT performed less efficiently than RBAT for smaller data sizes (8M and 16M). This is due to the overwhelming parallelization overhead, consisting of I/O, network and setup cost incurred by each MapReduce round. However, parallel RBAT grew sublinearly with regard to data sizes and grew much more slowly than RBAT did. When the data sizes were large, i.e. $|D|$ is between 64M and 256M, RBAT grew by a factor from 3 to 8, whereas our parallel RBAT only by a factor of 2 at maximum. In terms of response time, parallel RBAT performed up to 8 times faster than RBAT as data sizes increased from 64M to 256M.

TABLE III
Parameter Settings

Parameter	Default value
$\|D\|$	32M
$\|\Theta\|$	1,400
k	5
c	0.9

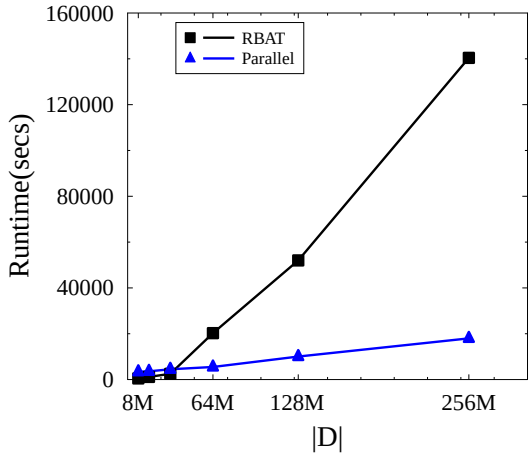

Fig. 3. Datasize vs. Runtime

We have also studied the scalability of parallel RBAT w.r.t. the number of computing nodes. Figures 5 and 6 show the response time and relative speed up, respectively, with varying number of computing nodes used in anonymization. The relative speedup is measured as the ratio of runtime obtained from the minimum cluster size used in our experiments to that obtained from a cluster size whose speedup is measured. We found that parallel RBAT scaled well when a small number of processing nodes were used. High throughput was obtained when utilising 8 computing nodes. Setting the cluster size to more than 8 nodes caused parallel RBAT to decrease its processing efficiency. We attribute this to the overhead caused by parallel configurations. That is, when 12 nodes were used, the computation on each node was reduced, making the overhead resulted from setting up parallel processing a

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

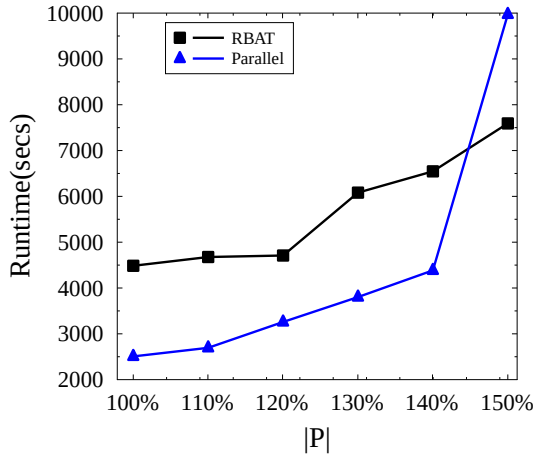

Fig. 4. Domainsize vs. Runtime

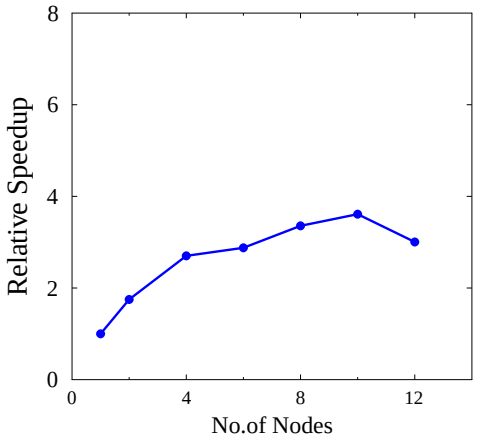

Fig. 6. No. of Processors vs. Relative Speedup

significant proportion of the overall response time.

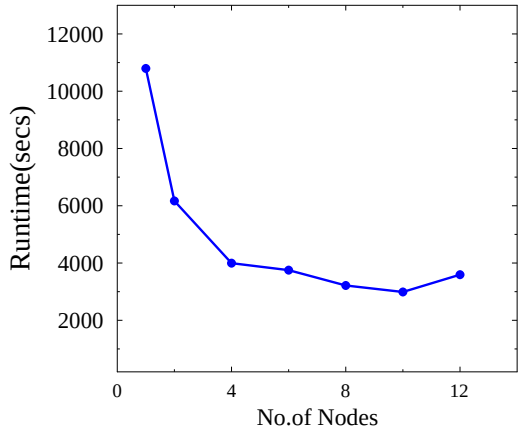

Fig. 5. No. of Processors vs. Runtime

Another important performance measure is scaleup [34], which captures the scalability of a parallel algorithm to handle larger datasets when more computing nodes are made available. We measured the scaleup of our parallel RBAT by the ratio of the time taken by a single processor to the time taken by m processors on a workload of $(m \times 8)$M transactions. We found that parallel RBAT had not made most effective use of resources, specifically when the cluster size was large. More specifically, we found that the main bottleneck lies in the split phase. The first step of the split phase is designed to use only one reducer. Therefore, increasing the number of available processors would increase the number of mappers used in anonymization, causing more intermediate output to be shuffled to the reducer, thereby incurring a high network cost. Also, splitting a generalized item $\tilde{i}$ requires $|\tilde{i}|-2$ MapReduce rounds. However, since data cannot be kept in memory of each mapper between any two rounds, parallel RBAT needs to reload the data partitions into mappers' memory at each round. For large datasets, the overhead of repetitive data loading increases, offsetting the gains from parallel computation.

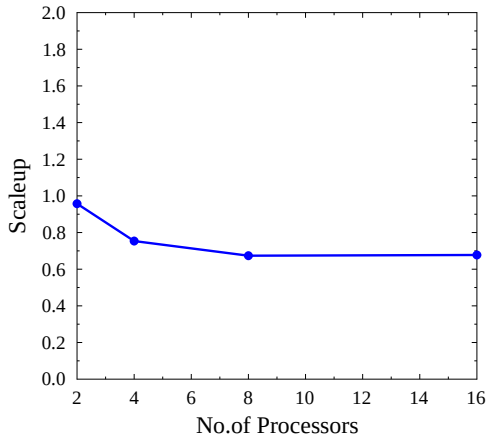

Fig. 7. No. of Processors vs. Scaleup

VI. Conclusions

In this paper, we studied how MapReduce may be used to improve the scalability of RBAT, a sequential method that has some desirable features for transaction anonymizaion. By partitioning the data, our parallel RBAT overcomes the limitation of requiring the whole dataset to fit into memory of a single machine, while providing significant speedup in anonymizing large datasets. Our empirical study using real-world transactions has shown that parallel RBAT can scale

sublinearly to large datasets of hundreds of millions of transactions. Our study has also shown that the overhead caused by configuring MapReduce rounds, and the cost of data loading and shuffling data over the network must be addressed, as it can easily offset the gains from parallel processing.

REFERENCES

[1] M. Terrovitis, N. Mamoulis, and P. Kalnis, "Privacy-preserving anonymization of set-valued data," *Proc. VLDB Endow.*, vol. 1, no. 1, pp. 115–125, Aug. 2008.

[2] M. Terrovitis, N. Mamoulis, J. Liagouris, and S. Skiadopoulos, "Privacy preservation by disassociation," *Proc. VLDB Endow.*, vol. 5, no. 10, pp. 944–955, Jun. 2012.

[3] Y. He and J. F. Naughton, "Anonymization of set-valued data via top-down, local generalization," *Proc. VLDB Endow.*, vol. 2, no. 1, pp. 934–945, Aug. 2009.

[4] J. Cao, P. Karras, C. Raïssi, and K.-L. Tan, "ρ-uncertainty: inference-proof transaction anonymization," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 1033–1044, 2010.

[5] Y. Xu, K. Wang, A. W.-C. Fu, and P. S. Yu, "Anonymizing transaction databases for publication," in *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '08, 2008, pp. 767–775.

[6] G. Loukides, A. Gkoulalas-Divanis, and J. Shao, "Anonymizing transaction data to eliminate sensitive inferences," in *Proceedings of the 21st International Conference on Database and Expert Systems Applications: Part I*, ser. DEXA'10, 2010, pp. 400–415.

[7] The Economist, "A special report on managing information: Data, data everywhere," *The Economist*, February.

[8] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[9] A. Gkoulalas-Divanis, G. Loukides, and J. Sun, "Publishing data from electronic health records while preserving privacy: A survey of algorithms," *Journal of Biomedical Informatics*, vol. 50, pp. 4–19, 2014.

[10] C. Aggarwal and P. Yu, *Privacy-Preserving Data Mining: Models and Algorithms*, ser. Advances in Database Systems. Springer US, 2008. [Online]. Available: https://books.google.co.uk/books?id=ndR8-wYsZKUC

[11] B. C. M. Fung, K. Wang, R. Chen, and P. S. Yu, "Privacy-preserving data publishing: A survey of recent developments," *ACM Comput. Surv.*, vol. 42, no. 4, pp. 14:1–14:53, Jun. 2010. [Online]. Available: http://doi.acm.org/10.1145/1749603.1749605

[12] M. Terrovitis, N. Mamoulis, and P. Kalnis, "Local and global recoding methods for anonymizing set-valued data," *The VLDB Journal*, vol. 20, no. 1, pp. 83–106, Feb. 2011.

[13] T. Iwuchukwu and 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*, ser. VLDB '07, 2007, pp. 746–757.

[14] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan, "Workload-aware anonymization techniques for large-scale datasets," *ACM Trans. Database Syst.*, vol. 33, no. 3, pp. 17:1–17:47, Sep. 2008.

[15] G. Loukides, A. Gkoulalas-Divanis, and J. Shao, "Efficient and flexible anonymization of transaction data," *Knowledge and information systems*, vol. 36, no. 1, pp. 153–210, 2013.

[16] C. Zhang, F. Li, and J. Jestes, "Efficient parallel knn joins for large data in mapreduce," in *Proceedings of the 15th International Conference on Extending Database Technology*, ser. EDBT '12, 2012, pp. 38–49.

[17] W. Lu, Y. Shen, S. Chen, and B. C. Ooi, "Efficient processing of k nearest neighbor joins using mapreduce," *Proc. VLDB Endow.*, vol. 5, no. 10, pp. 1016–1027, Jun. 2012.

[18] F. Afrati, A. Sarma, D. Menestrina, A. Parameswaran, and J. Ullman, "Fuzzy joins using mapreduce," in *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, April 2012, pp. 498–509.

[19] J. Huang, R. Zhang, R. Buyya, and J. Chen, "Melody-join: Efficient earth mover's distance similarity joins using mapreduce," in *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, March 2014, pp. 808–819.

[20] A. Ene, S. Im, and B. Moseley, "Fast clustering using mapreduce," in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '11, 2011, pp. 681–689.

[21] S. Papadimitriou and J. Sun, "Disco: Distributed co-clustering with map-reduce: A case study towards petabyte-scale end-to-end mining," in *Data Mining, 2008. ICDM '08. Eighth IEEE International Conference on*, Dec 2008, pp. 512–521.

[22] W. Zhao, H. Ma, and Q. He, "Parallel k-means clustering based on mapreduce," in *Cloud Computing*. Springer, 2009, pp. 674–679.

[23] R. L. Ferreira Cordeiro, C. Traina, Junior, A. J. Machado Traina, J. López, U. Kang, and C. Faloutsos, "Clustering very large multi-dimensional datasets with mapreduce," in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '11, 2011, pp. 690–698.

[24] M. Riondato, J. A. DeBrabant, R. Fonseca, and E. Upfal, "Parma: A parallel randomized algorithm for approximate association rules mining in mapreduce," in *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, ser. CIKM '12, 2012, pp. 85–94.

[25] I. Roy, S. T. V. Setty, A. Kilzer, V. Shmatikov, and E. Witchel, "Airavat: Security and privacy for mapreduce," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'10, 2010, pp. 20–20.

[26] C. Dwork, "Differential privacy," in *Encyclopedia of Cryptography and Security*. Springer, 2011, pp. 338–340.

[27] K. Zhang, X. Zhou, Y. Chen, X. Wang, and Y. Ruan, "Sedic: privacy-aware data intensive computing on hybrid clouds," in *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 2011, pp. 515–526.

[28] X. Zhang, C. Liu, S. Nepal, S. Pandey, and J. Chen, "A privacy leakage upper bound constraint-based approach for cost-effective privacy preserving of intermediate data sets in cloud," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 24, no. 6, pp. 1192–1202, June 2013.

[29] X. Zhang, C. Liu, S. Nepal, C. Yang, W. Dou, and J. Chen, "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.

[30] X. Zhang, L. Yang, C. Liu, and J. Chen, "A scalable two-phase top-down specialization approach for data anonymization using mapreduce on cloud," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 25, no. 2, pp. 363–373, Feb 2014.

[31] L. Sweeney, "k-anonymity: A model for protecting privacy," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 10, no. 05, pp. 557–570, 2002.

[32] K.-H. Lee, Y.-J. Lee, H. Choi, Y. D. Chung, and B. Moon, "Parallel data processing with mapreduce: A survey," *SIGMOD Rec.*, vol. 40, no. 4, pp. 11–20, Jan. 2012.

[33] G. Loukides, A. Gkoulalas-Divanis, and B. Malin, "COAT: Constraint-based anonymization of transactions," *Knowledge and Information Systems*, vol. 28.

[34] D. Taniar, C. H. Leung, W. Rahayu, and S. Goel, *High performance parallel database processing and grid databases*. John Wiley & Sons, 2008, vol. 67.