Sharma and Toshniwal *J Big Data* (2017) 4:4
DOI 10.1186/s40537-017-0064-9

Journal of Big Data

RESEARCH Open Access

Scalable two-phase co-occurring sensitive pattern hiding using MapReduce

Shivani Sharma* and Durga Toshniwal

*Correspondence:
shivani.vce@gmail.com
Department of Computer Science and Engineering, Indian Institute of Technology, Roorkee, Uttrakhand 247667, India

Abstract

Background: Expansion of Internet and its use for on-line activities such as E-Commerce and social networking are producing large volumes of transactional data. This huge data volume resulted from these activities facilitates the analysis and understanding of global trends and interesting patterns used for several decisive purposes. Analytics involved in these processes expose sensitive information present in these datasets, which is a serious privacy threat. To overcome this challenge, few sequential heuristics have been used in past where volumes of data were comparatively accommodating to these sequential heuristics; the current situation is not that much in-line and often results in high execution time. This new challenge of scalability paves a way for experimenting with Big Data approaches (e.g., MapReduce Framework). We have agglomerated the MapReduce framework with adopted heuristics to overcome this challenge of scalability along with much-needed privacy preservation and yields efficient analytic results within bounded execution times.

Methods: MapReduce is a parallel programming framework [16] which provides us the opportunity to leverage largely distributed resources to deal with the Big Data analytics. MapReduce allows the resource of a largely distributed system to be utilized in a parallel fashion. The simplicity and high fault-tolerance are the key features which make MapReduce a promising framework. Therefore, we have proposed a two-phase MapReduce version of these adopted heuristics. MapReduce framework divides the whole data into 'n' number of data chunks D = {d 1 d U 2 U d 3 ..... U d n } and distributes them over 'n' computing nodes to achieve the parallelization. The first phase of MapReduce job runs on each data chunk in order to generate intermediate results, which are further sorted and merged in the second phase to generate final sanitized dataset.

Results: We conducted three set of experiments, each with five different scenarios corresponding to the different cluster sizes i.e., n = 1,2,3,4,5 where 'n' is a number of computing nodes. We compared the approaches with respect to real as well as synthetically generated large datasets. For varying data sizes and varying number of computing nodes, it has been observed that sanitization time required by the MapReduce-based algorithm for same size dataset is much less than the sequential traditional approach. Further, the scalability can be improved by using more number of computing nodes. Lastly, another set of experiments explores the change in sanitization time with varying sizes of the sensitive content present in a dataset. We evaluated the effectiveness of proposed approach in different scenarios, with varying cluster size from 1 to 5 nodes. It has been observed that still the execution time of our approach is much less than traditional schemes. Further, no hiding failure, artifactual patterns

have been observed during the experiments as well as in terms of misses cost also the MapReduce version performance is same as of traditional approaches.

Conclusion: Traditional approaches for data hiding primarily MaxFIA and SWA were lacking with due inability to tackle large voluminous data. To subjugate the new challenge of scalability, we have implemented these basic heuristics with Big Data approach i.e., MapReduce framework. Quantitative evaluations have shown that the fusion of MapReduce framework with these adopted heuristics fulfills its obligatory responsibility of being scalable and many-fold faster for yielding efficient analytic results.

Keywords: Privacy preservation, MapReduce framework, Big data, Information hiding

Background

With advancements in technology, applications such as social networking, e-commerce, wireless sensors etc. tend to produce a large volume of data. These voluminous datasets facilitate the analysis and understanding of much needed global trends and interesting patterns, for which organizations/clients may require to share their data with others. Sharing may cause exposure of sensitive information present in these datasets and might invite number of privacy threats [2] (e.g. medical records or financial records if mined can provide significant human benefits but the failure of privacy might allow malicious users or providers to misuse this information which can cause considerable economic or social loss).

A number of privacy preservation techniques exist some of which focus on preserving the privacy of outsourced data i.e. for the secure data storage and computation at third party [1, 3, 4]. Methods such as anonymization, encryption used in these techniques, preserve the privacy by considering whole data as private. These approaches fairly deal with the issues like frequency attack, access control, etc. but the analytically generated rules/patterns still can reveal sensitive information present in these datasets. This is due the inability of these techniques which do not obstruct the mining of patterns exposing any sensitive information. Therefore, we need to restrict the generation of such sensitive rules, before sharing or analyzing data.

A number of data hiding techniques are used in [9–14] to mask sensitive knowledge before sharing or analyzing datasets. Existing data hiding techniques can be broadly divided into three categories: heuristic, border and exact. Heuristics based approaches are simple and provide high privacy level but process the data in sequential fashion [13]. These heuristics are fairly adequate for small/medium sized datasets though the current situation is not that much in-line. The exponential increase in data volume and sequential nature of conventional data hiding techniques often result in high execution time or sometimes even non-feasible. This new challenge of scalability paves a way of experimenting with Big data approaches (e.g. MapReduce framework) for parallel processing.

MapReduce parallel programming framework [8] provides abundance of computation and storage power and can be seen as a promising solution for Big Data analysis. MapReduce framework agglomerated with adopted heuristics overcomes the challenge of scalability along with much-needed privacy preservation and yields efficient analytic results for real execution time. Key features like flexibility, simplicity and fault-tolerance make MapReduce Framework a significant choice [17]. Further, MapReduce excels at

distributing heavy computational operations across a cluster of distributed computing machines while abstracting many of the underlying implementation details (e.g. load balancing, job monitoring, data partitioning etc) [19].

In our work, we propose a scalable and fast heuristic-based approach to hide sensitive knowledge using MapReduce Framework. To adequately utilize the abundant power of the framework, the sanitization process is split into two MapReduce phases. Initially, original data is partitioned into 'n' number of data chunks which are distributed over 'n' computing machines. On each data chunk, a subroutine runs to select the victim item against each sensitive itemset and produces an intermediate result. In the second phase, the corresponding victim item is removed from identified transaction. Further, all these modified transactions are sorted and combined to obtain final sanitized dataset which can be uploaded or shared with others. We deliberately designed a number of MapReduce jobs to collaboratively mask the sensitive knowledge. We evaluated our approach on large-scale real transactional dataset as well as synthetically generated datasets using IBM Quest Synthetic Data Generator. Results demonstrate that our approach is significantly efficient and scalable over existing data hiding techniques.

Our work contributed in three major directions. Firstly, we designed MapReduce jobs to mask sensitive knowledge in highly scalable fashion. Secondly, we proposed some modifications in basic heuristics to prevent over-hiding and high communication/computation cost while collaborating them with MapReduce Framework. Parallelization ensures to sanitize large-scale dataset in real execution time Lastly, with reproducible quantitative evaluations, we have shown that the MapReduce framework combined with basic heuristics overcome the identified challenges in significant and efficient manner.

The rest of this paper is organized as follows. "Related work and problem analysis" section briefly discusses some related work. "MapReduce" section discusses the basics of the MapReduce framework. In "Proposed MapReduce version of MaxFIA and SWA" section, scalable two-phase co-occurring sensitive pattern hiding heuristic approach is described in detail. "Experiments and performance analysis" section presents experimental results and performance analysis. Finally, "Conclusion" section draws the conclusion.

Related work and problem analysis

Related work

The issue of scalability and high execution time has been widely investigated and resolved mainly for outsourced techniques like Xuyun et al. identified issue of scalability during anonymization of large-scale dataset [1]. They introduced scalable two-phase top-down anonymization approach by using MapReduce framework over a cloud. The primary TDS approach introduced by Fung et al. [18] has been divided into multiple MapReduce jobs which collaboratively anonymize large scale data in high scalable manner. EFPA is a privacy preserving approach [4] for association rules on a cloud. A number of parties can combine their data and mine association rule with no data leakage. EFPA used encryption and perform mining over encrypted data only. This ensures privacy preservation with low computational cost. Hybrid cloud (public and private) architecture is introduced in [5] to ensure the privacy of data before making it accessible to others. The private cloud has been provided as an interface to access a public cloud. The data utilization system and private cloud are used to encrypt data. The system provides security by outsourcing the

cryptographic access control mechanism. The system claims reduced computational cost at user side. Yi et al. [3] introduced another cryptographic approach for preserving the privacy of association rules in a cloud. The ElGamal cryptographic approach has been used in distributed fashion where semi-honest server mine the encrypted data. Liu et al. [2] proposed privacy preserved scanning of big data using MapReduce framework. The technique minimizes the sensitive data exposure during the data detection for outsourcing data securely. Wei et al. [6] proposed a secure computation auditing protocol which bridges secure storage and computation within a cloud and achieves privacy using verifier signature, batch verification, and probabilistic sampling techniques. Yan et al. [7] proposed two privacy preserving techniques for trust evaluation based on additive homomorphic encryption. It is applicative approach and supports big data process.

Most of the work has been done for handling scalability issue while preserving data privacy for outsourced data i.e. for secure data storage and computation at a third party. But for sanitization techniques like hiding sensitive co-occurring patterns or masking restricted rules, scalability is still an issue. Masking techniques play a crucial role in privacy preservation as even after anonymization or encryption, mined rules can generate certain patterns which may expose sensitive information. Hence, it is equally important for masking techniques to be scalable and fast enough such that data privacy can be preserved in both ways. Further, we will discuss some primary data hiding techniques and their limitations to understand the background of the problem.

Traditional heuristic approaches

A number of sequential heuristics are used to preserve the data privacy by hiding sensitive co-occurring patterns. Atallah et al. [11] in proposed a primary heuristic technique to hide the sensitive association rules by reducing the support of their generating itemsets. The authors proposed the construction of lattice-like graph in the database through which greedy iterative traversal is made for identifying and hiding maximum frequent item related to the sensitive rule. All the sensitive rules are masked in one by one fashion. Dasseni et al. [14] generalized the problem and proposed three ideas to hide sensitive frequent itemsets as well as sensitive rules. The first two schemes reduce the confidence of sensitive rule either by increasing the antecedent support or by decreasing the frequency of rule consequent. The third strategy decreases the support of either the antecedent or consequent of the rule but not both till the confidence of the rule becomes less than the minimum threshold. The technique is based on an assumption that the items appearing in one sensitive itemset will not appear in another. Verykios et al. [15] extended the work in [14] and tried to improve the data quality by hiding the maximum support victim item from the identified transaction exhibiting minimum length. Oliveira et al. introduced multiple rules hiding approach in [9] which requires two database scan without any concern of the number of sensitive itemsets need to be masked. Authors introduced three ways to select victim item i.e. MinFIA, MaxFIA, and IGA. MinFIA identifies minimum support item as a victim and removes it from the supporting transaction. MaxFIA chooses a maximum support item as the victim. Lastly, IGA is a hybrid approach which clusters the sensitive patterns sharing same itemsets and hides the whole cluster at once. SWA [10] is an improved version of [9] as it requires single database scan and aims to hide all the restrictive patterns in five easy steps. It is simple

Table 1 Summary of existing traditional heuristic based data hiding techniques

S. no	Approach	Technique used	Achieved	Issues
1	[11]	Constructed lattice-like graph of a dataset Greedy iterative traversal to immediate subset Selected victim with maximum support	Good privacy level Simple and fast	Have not considered the extent of loss of support for large itemsets hence data quality get affected Scalability to handle large-scale data
2	[14]	Increase support of antecedent $(A) \rightarrow B$ Decrease support of consequent $A \rightarrow (B)$ Hybrid decrement till confidence or support goes below the threshold	Decrease the support or confidence but not both	Based on strong assumption of any item contained in one sensitive itemset will not appear in another sensitive itemset Scalability
3	[15]	Deleted maximum support item $i \in s$ from minimum length transaction The second algorithm sort sensitive itemset in terms of size and support of itemsets and mask them in round robin fashion	Sanitize minimum length transaction first in order to decrease the side effect on non-sensitive data Second algorithm is fair enough by masking in round robin fashion	Scalability is still an issue High execution time for large dataset
4	[9]	MaxFIA: deleted maximum support item $i \in s$ where $s \subseteq T$ MinFIA: Delete minimum support item $i \in s$ where $s \subseteq T$ IGA: make clusters of sensitive patterns sharing same itemsets and delete max or min support item	Cluster formation hides the set of sensitive itemsets at once No traversal required, easily count and select max and min support item Sensitive dataset is separated out in order to reduce the data size and sanitization time	Issue of scalability and high execution time in case of large scale dataset
5	[10]	SWA mask sensitive rules by hiding maximum frequency item $i \in s$ where $s \subseteq T$ SWA requires single database scan	Conceal all the sensitive rules Require single database scan Sliding window concept made the approach scalable to some extent	High execution time as well as scalability when data is big data
6	[12]	Aggregate: deleted transaction $T \cap S$ supporting maximum sensitive itemsets Disaggregate: delete maximum support item $i \in s$ where $s \subseteq T$ from remaining transactions	Hybrid approach is fast as it selectively identify transaction and delete maximum support item	Direct deletion of transactions affects the data quality as the transactions may contain non-sensitive information too Scalability issue exists

sliding window approach which maintains good data privacy and scalability. Amiri [12] claims to provide a high data quality and lower distortion by using the aggregation and disaggregation scheme. The author proposes to delete the transactions supporting the maximum number of sensitive itemsets directly and further, delete the maximum support victim from the remaining sensitive transactions. Direct deletion of transactions may hide the sensitive itemsets in much less time but, possibly degrades the data quality. Because these deleted transactions may also contain some non-sensitive information. A summary of all the above traditional techniques is presented in Table 1.

Problem analysis

From the literature discussed in "Related work" and "Traditional heuristic approaches" sections, it can be stated that we need to preserve the data privacy in both ways i.e. for secure data storage/rule mining at a third party as well as by masking restricted sensitive information before data sharing. A number of traditional techniques exist in both categories which perform well with comparatively accommodating volumes of data; the current situation is not that much in-line and often results in high execution time and sometimes result in non-feasibility. Much work has been done to improvise outsourced techniques but data hiding techniques are still struggling with these limitations.

From Table 1, it can be observed that both MaxFIA [9] and SWA [10] are primary data hiding techniques offering multiple advantages over others in terms of simplicity, required number of database scans etc. Both these approaches identify maximum support item as a victim and remove it from the supporting transactions. For accommodating volumes of data, as discussed in "Traditional heuristic approaches" section, MaxFIA is a novel scheme but when applied to a big dataset, it results in high execution time and non-scalability. Further, K-sliding window approach used in SWA resolves the scalability issue but these K-data windows still need to be processed in a sequential manner i.e. one after another which again results in high execution time.

These new challenges of scalability and high execution time paves a way for experimenting with Big Data approaches (e.g. MapReduce framework). Thus, here we propose parallelized version of these conventional approaches by using MapReduce Framework.

Challenges in collaborating basic heuristics with MapReduce

Conventional MaxFIA and SWA techniques maintain a list of supporting transaction Ids against each sensitive itemset [9, 10]. After every victim item removal, a look ahead procedure is performed to verify if the transaction has also been selected for other restrictive rules. If yes, and the victim item we removed is also the part of this other restricted rule, we need to remove the transaction from the respective list. This improves the misses cost but requires to repeat the procedure for every transaction and sensitive itemset; it is a time consuming, computationally expensive and sometimes infeasible procedure for Big Data and a parallel environment. Therefore, these heuristics require to be modified in terms of following:

- Traditionally all sensitive itemsets are masked in one by one fashion.
- After each victim item removal,transaction list against every affected sensitive itemset is revised. In a parallel environment, the lookahead procedure requires all nodes

to communicate and revise their set of sensitive itemset and respective list, which ultimately will increase communication and computation cost.

- If every node sanitizes their set of transaction (data chunk) independently then there is a chance of over-hiding which will degrade the data quality of sanitized dataset.
- To study how to divide a dataset into small data chunks such that even during parallel sanitization, the minimum length/DoC transaction can be modified first.

Therefore, we require few modifications of these basic heuristics to handle these challenges while implementing them in a parallel environment (i.e. MapReduce framework). The proposed improved version maintains a global index file with the master node containing sensitive itemset, supporting transaction Ids list and their delta value to keep a check if a particular sensitive itemset further needs to be masked or not. This methodology ultimately prevents over-hiding. Transaction ids list will now be used to find if a restricted pattern belongs to a transaction or not i.e. by directly searching for the particular transaction id in the corresponding list. This helps in speeding up the sanitization process. Further, the global index file also mitigates the requirement of revising transaction list recursively during look ahead procedure. Now all computing nodes need to propagate the effect of victim item removal directly to the global file by reducing the corresponding delta value. This methodology is neither computationally expensive nor requires any communication among computing nodes but only with the master node. Lastly, for maintaining the requirement of sanitizing minimum length/DoC transaction first, we have modified data partitioner explained in "Overview" section. These small amendments in adopted heuristics when combined with MapReduce framework ultimately resolve all identified issues and help in masking sensitive itemsets in a parallel fashion within the real execution time.

MapReduce

MapReduce is a parallel programming framework [16] which provides us the opportunity to leverage largely distributed resources to deal with the Big Data analytics. The framework divides and distributes the Big Data as well as heavy computation involved,

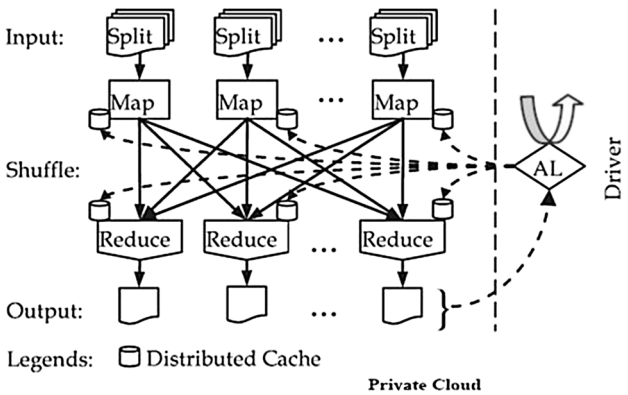

Fig. 1 Computation phase of MapReduce [1]

over n computing machines. These 'n' computing machines are combined to form a cluster. User need to specify two functions i.e. Map and Reduce which accept and process dataset in form of (key, value) pair and output the processed data again in (key value) same format i.e. $(key_1, val_1) \rightarrow (key_2, val_2)$. Map function processes the data and generates intermediate (key_2, val_2) pair which is given as the input to the reducer function for merging the values associated with the same key. MapReduce allows the resource of a largely distributed system to be utilized in a parallel fashion. The simplicity and high fault-tolerance are the key features which make MapReduce a promising framework. It hides the complications and handles failure automatically. Simple MapReduce computation phase can be seen in Fig. 1 MapReduce provides two level of parallelization i.e. task level and job level. When a number of MapReduce jobs execute together, it is called job level parallelization and if multiple mappers and reducers run within a single job, it refers to task level parallelization.and distributes them

Proposed MapReduce version of MaxFIA and SWA

MapReduce framework divides the whole data into 'n' number of data chunks $D = \{d_1 \cup d_2 \cup d_3 \cdots \cup d_n\}$ and distributes them over 'n' computing nodes. This is called *data partition*. By default, the maximum size of each data chunk is 64 MB depending on which value of n varies. We have deliberately designed a number of Map Reduce jobs which processes each data chunk parallel in two MapReduce phases.

Overview

Proposed two-phase MapReduce version can be viewed as a composition of some easy and small objectives. The first phase of MapReduce job runs on each data chunk in order to generate intermediate results, which are further sorted and merged in the second phase to generate final sanitized dataset. The overview of these phases is discussed below:

- **Phase-I**
- Sub-routine 1: (*Data separator*) Sensitive transactions (T*) $\{if \quad s \subseteq T, where \quad s \in S, T \in D\}$ are separated out from the non-sensitive transactions (T') in order to reduce the size of dataset need to be processed further.
- Sub-routine 2: (*Frequency calculator*) Support of each 1-frequent item calculates and stores in an support index file(SIF), such that support of any item can be directly accessed by any node in a cluster.
- Sub-routine 3: (*Victim identifier*) Against each sensitive itemset, an item with maximum support is selected as a victim.
- Sub-routine 4: (*Transaction sorting*) Sensitive transactions are sorted depending on the given condition (e.g. Length of transaction, Degree of conflict etc.).
- Sub-routine 5: (*Sensitive itemset effect calculator*) It calculates $\Delta = Current \quad support - Th(minimum \quad support)$ the minimum number of times

sensitive itemset need to be masked for making it infrequent. All the sensitive itemsets and their corresponding Δ values are stored in a global index file (GIF).

- **Phase-II**

Data partitioner: Initially, the sensitive dataset determined in Phase-I is sorted in increasing order of length/DoC. A total number of computing nodes (n) in a cluster and sorted sensitive dataset is provided as input to the partitioner, where 'n' new buckets(sets) are initialized. Every (i+n)th transaction is provided to the (i)th bucket. Further, these data buckets can be divided according to the value set for the size of each data chunk.

- Sensitive transactions are further divided into data chunks using data partitioner and distributed over n computing machines.
- At each node, victim item against each sensitive itemset is removed till all restricted information is hidden.

Finally, modified transactions and a non-sensitive set of transactions are combined to form final sanitized dataset. Further, we will take the techniques one by one and discuss their MapReduce version in detail.

MapReduce version of MaxFIA

Here, we introduce two-phase MapReduce version of MaxFIA which selects victim item with maximum support and sanitizes the transaction with a minimum degree of conflict (DoC) first. Initially, we deliberately design the subroutine which runs over each of the partitioned datasets in parallel. For each restricted sensitive itemset, subroutine 1: concretely computes the frequency of each 1-frequent item ($item$, 1), subroutine 2: computes the DoC for each transaction i.e. number of the sensitive itemset, the transaction is supporting (T_{id}, 1). Further, subroutine 3: computes Delta value Δ for each sensitive itemset s. MapReduce framework sorts, shuffles, and merges these key value pairs to form a value list against each key i.e. (T_{id}, $count_{list}$), ($item$, $count_{list}$) and (s, Δ). These lists are provided as input to the reducers where global item support, global sensitive itemset support and transaction DoC are calculated with respect to the whole database. Transactions with DoC > 0 are called Sensitive Transactions and others supporting none of the sensitive itemsets are called Non-Sensitive Transactions, which are directly merged with final sanitized dataset. A group of sensitive transactions is sorted in ascending order of DoC such that transaction supporting minimum number of the sensitive itemset is sanitized first, to reduce the side effect on non-sensitive information. The item with the maximum support is selected as the victim item, which is removed from the identified transaction. The pseudocode of Phase I is given in Algorithm 1.

Algorithm 1 MaxFIA Phase I- Map & Reduce

procedure PHASE 1: MR
  Input: Data chunks db_i and Set of sensitive itemsets S
  Output: Victim item v and Set of sensitive transactions TB^*
  Map:
  Initially, a counter is set to calculate the DoC for each transaction
  for Each transaction $T \in TB^*$ **do**
    for Each sensitive itemset $s \in S$ **do**
      if $s \subseteq T$ **then**
        Output pair(s, T_{id})
        Output pair$(s, 1)$
        Increase the value of counter++
        for Each item $i \in s$ **do**
          if item $i \subseteq T$ **then**
            Output pair(item(i), 1)
    if Counter > 0 **then**
      Output pair$(T_{id}, counter)$
  Reducer:
  for Each key value pair of 1-frequent item $(item(i), 1)$ **do**
    Add the total support each item i is exhibiting
    Output pair$((item(i), total_{count}) -> SIF)$
  for Each sensitive itemset $s \in S$ **do**
    Merge the list to T_{id} corresponding to the same key
    Output $(s, List_{T_{ids'}}, \Delta) -> GIF$
    The sensitive itemset with corresponding T_{id} and Δ value is stored in global index file
  for Each key value pair of transaction and DoC $(T_{id}, counter)$ **do**
    Merge and sort the transactions list against each sensitive itemset in ascending order of the degree of conflict)
  An item with maximum support is selected as victim item.

In phase II, set of sensitive itemsets, corresponding victim item, a chunk of sensitive transactions are provided as input to the mapper. For each sensitive itemset, the corresponding victim item is removed consecutively in distributed fashion from the list of identified transaction $\{(T - v), \quad where \quad v \in \{s, T\}, s \in S \quad and \quad s \subseteq T\}$ with lowest DoC. After every victim removal, *Delta* and support of item(i) is modified directly in GIF and SIF. Finally, the obtained key value pair from the mapper will be given as the input to the reducer, which merges the sanitized transaction and the non-sensitive transactions obtained in Phase-I together in order to generate fully sanitized dataset which can be shared and analyzed with much-needed privacy. The pseudocode for Phase-II mapper and reducer is given in Algorithm 2.

Algorithm 2 MaxFIA Phase II- Map & Reduce

procedure PHASE 1: MR
  GIF anf SIF is provided at the global level so that all thye computing machines have access to it
  Input: Sensitive data chunk d_i^*, Victim item v corresponding to each sensitive item $s \in S$
  Output: Final Sanitized dataset
  Map:
  for Each transaction T $\in d_i^*$ **do**
    for Each sensitive itemset $s \in S$ **do**
      if $s \subseteq T \& \Delta(s) > 0$ **then**
        Delete the victim item corresponding to sensitive itemset s ie. $\{T^* = (T - v), where \quad v \in \{s, T\}, s \in S \quad and \quad s \subseteq T\}$
        Reduce the Δ for (s) in GIF
        Reduce the support of (v) in SIF
      Output pair $(T_{id}, T*)$
  Reducer:
  Merge set sanitized transactions D^* with the set non-sensitive transaction D' to generate fully sanitized privacy preserved dataset

These two phases explore the abundant computation power of the MapReduce parallel programming framework and can handle even voluminous data by using largely distributed computing nodes. It is shown in "Experiments and performance analysis" section, that MapReduce can process the huge data volume in highly scalable fashion and within bounded execution time.

MapReduce version of SWA

SWA is an improvised version of MaxFIA and requires only single database scan. Sliding window approach used in SWA make it quite scalable but, still in case of big data the sanitization time is huge. Hence, MapReduce implementation helps the approach to be fast enough such that even voluminous data can be sanitized in an adequate amount of time. Unlike MaxFIA, SWA sorts the identified supporting transactions against any sensitive itemset in increasing order of their 'length' to minimize the side effects on non-sensitive information. The data is processed by considering the set of transactions within K-size window. It adds to scalability but still each data window needs to be processed one after another i.e. in a sequential fashion and often results in high execution time. MapReduce framework not only reduces the sanitization time but also mitigates the requirement of sliding window approach. This is possible since it directly divides data into small chunks and sanitizes them in parallel.

In Phase-I, for each sensitive itemset $s \in S$, the frequency of a 1-frequent item is calculated by a subroutine in mapper and intermediate key-value pair (*item*, 1) is generated. Subroutine 2: separates the sensitive transactions from the set of non-sensitive ones. Subroutine 3: calculates the length of the transaction denoted by variable 'len' i.e. number of total items in the transaction (T_{id}, *len*) and further, subroutine 4: evaluates the Δ value for each sensitive itemset. The reducer processes these intermediate results and identifies the victim item against each sensitive itemset. The set of sensitive transactions gets sorted in ascending order of length 'len' such that the minimum length transaction is sanitized first. Two global files SIF and GIF storing item support and Δ value for each sensitive itemset are created respectively. The pseudocode of Phase I is given in Algorithm 3.

Algorithm 3 SWA Phase I- Map & Reduce

procedure PHASE 1: MR
  Input: Data chunks db_i and Set of sensitive itemsets S
  Output: Victim item v and Set of sorted sensitive transactions TB*
  Map:
  Variable len and counter are initialized
  for Each transaction $T \in TB^*$ **do**
    for Each itemset $I \in T$ **do**
      Increase the value of variable len ++
  for Each transaction $T \in TB^*$ **do**
    for Each sensitive itemset $s \in S$ **do**
      if $s \subseteq T$ **then**
        Output pair (s, T_{id})
        Output pair $(s, 1)$
        Increase the value of counter++
        for Each item $i \in s$ **do**
          if item $i \subseteq T$ **then**
            Output pair(item(i), 1)
        if Counter > 0 **then**
          Output pair(T_{id}, len)
  Reducer:
  for Each key value pair of 1-frequent item (item(i), 1) **do**
    Add the total support each item i is exhibiting
    Output pair$((item(i), total_{count}) -> SIF)$
  The key value pair with the same key is given to the same reducer to calculate Δ *of each s*
  for Each pair with same key, s **do**
    Merge the values with the same key and prepare Transaction's id list against each sensitive itemset s
    Output $(s, List_{T_{ids'}}, \Delta) -> GIF$
  An item with maximum support is selected as victim item.
  Sort the transactions in each list against each sensitive itemset s in increasing order of their length 'len' by considering the pair (T_{id}, len) *provided to each reducer*

In Phase-II, set of the sensitive itemsets, corresponding victim item and sorted list (in terms of length) of sensitive transactions are provided as input to the mapper. Each node masks sensitive itemsets in parallel by removing the victim item. Finally, sanitized transactions are combined with the set of non-sensitive transactions to form final sanitized dataset, which can be shared with other parties. The pseudo-code of Phase-II is given in Algorithm 2. The only difference in Phase-II for both schemes is in the way of sorting taking place. In MaxFIA, transaction batch is sorted on the basis of DoC and in SWA it is sorted on the basis of length respectively.

Experiments and performance analysis

With reproducible quantitative evaluations, we have shown that MapReduce framework agglomerated with adopted heuristics, overcomes this challenge of scalability along with much-needed privacy preservation and yields efficient analytic results within bounded execution times. We implemented conventional heuristics and the proposed approach in Hadoop using JAVA. Hadoop is an open source software system implementing MapReduce. We deployed the above approaches in a local cluster of five nodes with one master and rest as slaves. We conducted three set of experiments, each with five different scenarios corresponding to the different cluster size i.e. n = 1, 2, 3, 4, 5 where 'n' is a number of computing nodes. We compared the approaches with respect to real as well as synthetically generated large datasets. The size of dataset varies from 10 to 25 GB [10.8, 15.4, 21.3, 24.9]. Dataset with size 21.3 GB is real transactional data obtained from

"kaggel's- dataset repository" [20] and others are synthetically generated using IBM Quest Synthetic Data Generator.

First, experimental setup compares both proposed and existing version of MaxFIA and SWA from the perspective of scalability and sanitization time. Figures 2 and 3 show the change in execution time with varying data size i.e. ranging from [10 to 25 GB]. It can be clearly observed that sanitization time required by the MapReduce-based algorithm for same size dataset is much less than the sequential traditional approach. Therefore, it can be concluded that even if we implement our approach on single node, we can observe that the sanitization time is much less compared to traditional approaches. This is because even on single node setup, Hadoop provides two mappers running in parallel.

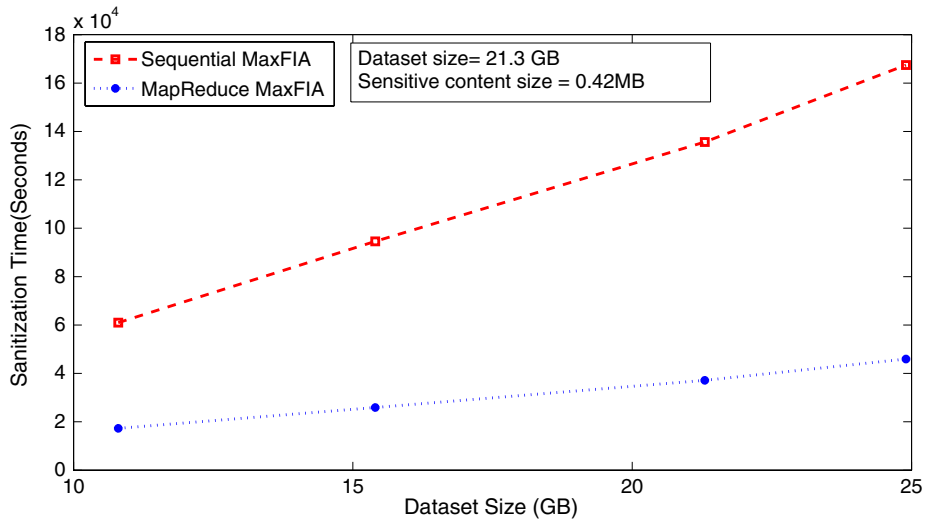

Fig. 2 Effect of varying data size on sanitization time with single computing node (MaxFIA)

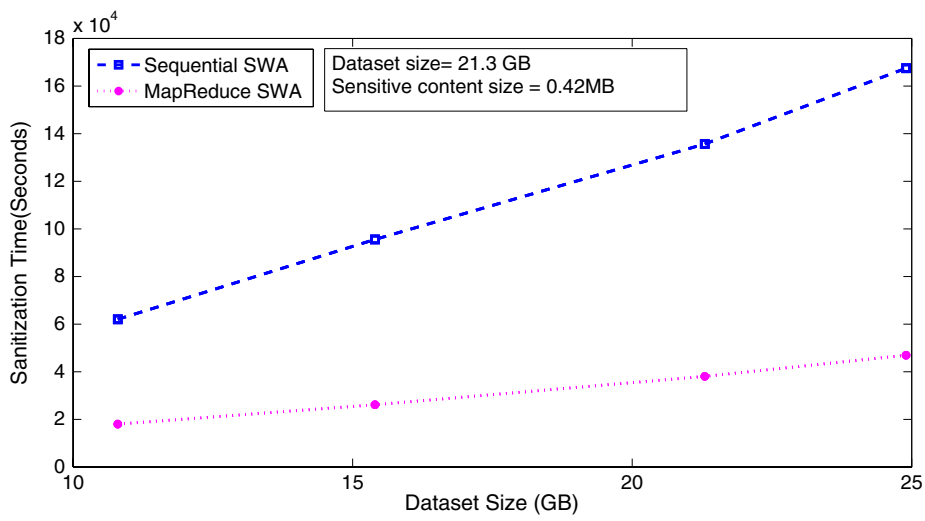

Fig. 3 Effect of varying data size on sanitization time with single computing node (SWA)

In the second set of experiments, we evaluated the effectiveness of MapReduce version in terms of change in execution time with varying data and cluster size. Figures 4 and 5 show the comparison of both approaches with respect to data size ranging from [10 to 25 GB] and a number of nodes within the cluster varying from 1 to 5. It can be observed that with the increase in the number of computing nodes, the execution time decreases for both schemes, because of the parallelization.

Lastly, the third set of experiments explores the change in sanitization time with varying size of the sensitive content (0.5–2 MB) which exist in the dataset of size 21.3 GB. We evaluated the effectiveness of proposed approach in different scenarios, with varying cluster size from 1 to 5 nodes. Figures 6 and 7 compare the execution time for both the

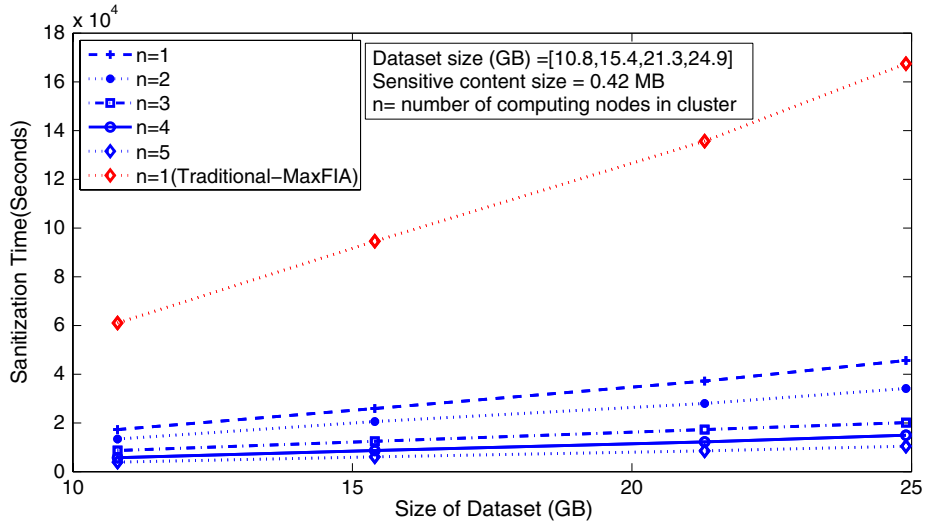

Fig. 4 Effect of varying data size on sanitization time with varying number of computing nodes in cluster (MaxFIA)

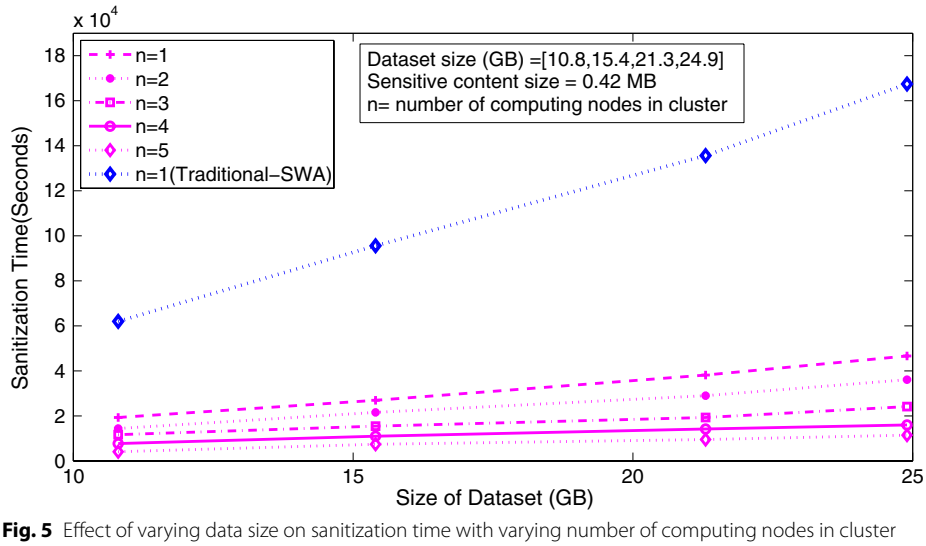

Fig. 5 Effect of varying data size on sanitization time with varying number of computing nodes in cluster (SWA)

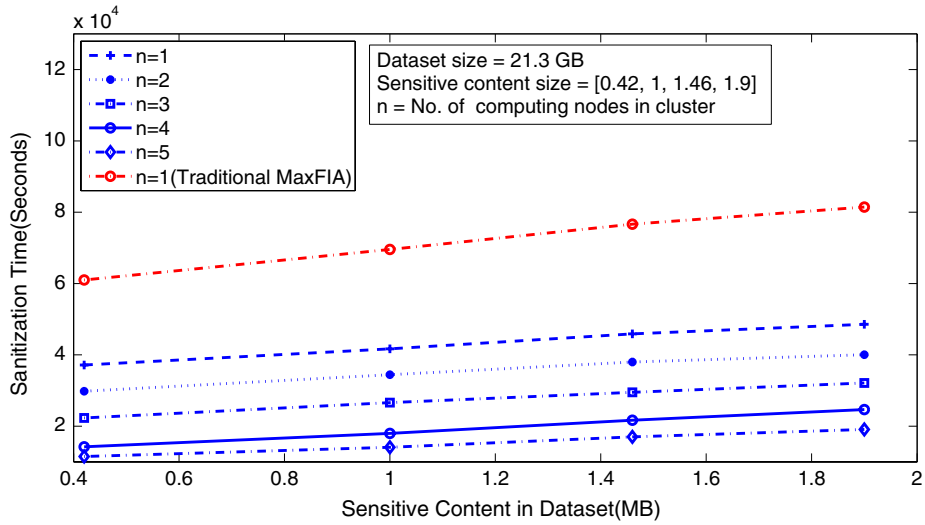

Fig. 6 Effect of varying sensitive content size present in dataset with varying number of computing nodes in cluster (MaxFIA)

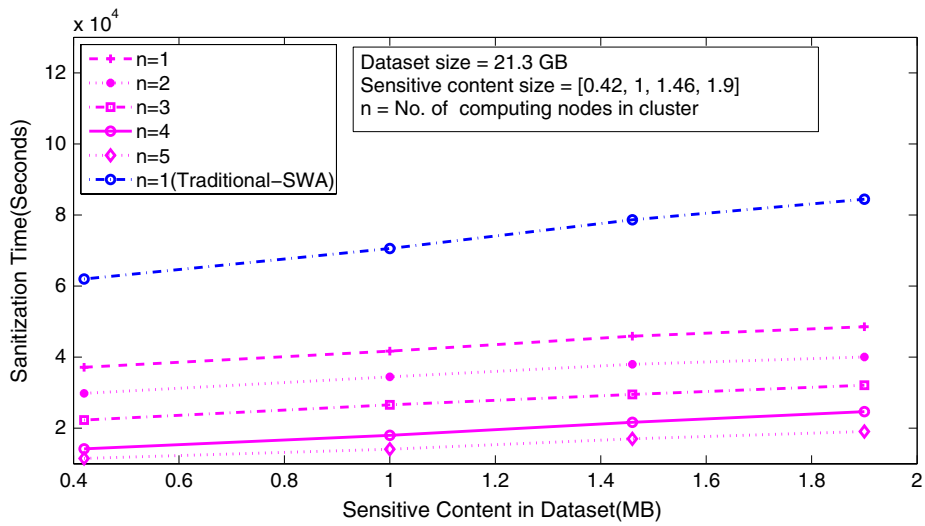

Fig. 7 Effect of varying sensitive content size present in dataset with varying number of computing nodes in cluster (SWA)

approaches. It can be clearly observed that with the increase in sensitive content size the sanitization time increases. But still, the execution time of our approach is much less than traditional schemes. Further, with the increase in the number of computing nodes, sanitization time can be further reduced.

The efficiency of the proposed method in terms of privacy preservation can be explained using performance measures introduced in [10]: *Hiding Failure* There will be no hiding failure as the approach is committed to run till all the sensitive itemsets are masked. *Artifactual Patterns* As no foreign element has been added to the dataset for masking sensitive content, therefore no artificial patterns are expected to be generated. Lastly, in terms of *Misses Cost:* the performance of parallelized and original heuristics

is expected to be the same because the quality of information hiding depends on two major factors that are- victim item selection and transaction selection. In either way (parallelized or traditional approach), maximum support item is selected as victim item and transactions are sanitized in the increasing order of their length or DoC. Hence, it can be clearly deduced that the privacy level achieved by both traditional and parallel version will be approximately same.

Finally, we can conclude that the MapReduce version of data hiding techniques outperforms the existing approaches in terms of scalability and execution time. Further, the efficiency of proposed approach can be improved by engaging more number of computing nodes in a cluster. MapReduce framework supports cloud environment and hence, our approach can be easily implemented on a cloud infrastructure.

Conclusion

Expansion of Internet and its use for on-line activities (e.g. social networking, e-commerce) overwhelmed E-Business with huge data volume, which facilitates organizations for analyzing and understanding global trends and patterns. Sharing and analysis involved may expose personal or confidential information a dataset may contain; which is certainly a serious privacy threat. Traditional approaches for data hiding primarily MaxFIA and SWA were lacking with due inability to tackle large voluminous data. To subjugate the new challenge of scalability we have implemented these basic heuristics with Big Data approach i.e. MapReduce framework. Quantitative evaluations have shown that the fusion of MapReduce framework with these adopted heuristics fulfills its obligatory responsibility of being scalable and many-fold faster for yielding efficient analytic results.

Authors' contributions

DT contributed to the underlying idea and helped in drafting the manuscript. DT played a pivotal role in guiding and supervising throughout. SS developed and implemented the idea, designed the experiments, analyzed the results and wrote the manuscript. Both authors read and approved the final manuscript.

Acknowledgements

We are thankful to DeitY for providing the scholarship to do the research in Ph.D. Program.

Competing interests

The authors declare that they have no competing interests.

Appendix

s	Sensitive itemset
v	Victim item selected to remove in order to reduce support of corresponding sensitive itemset
S	Set of sensitive itemsets
D	Dataset
T	Transaction 2 D
T*	Sensitive transaction i.e. one which supports one or more sensitive itemsets
DOC	Degree of conflict i.e. number of sensitive itemsets transaction supports
len	Length of transaction i.e. total number of items transaction consists of
TB*	Set of sensitive transactions

db	Data chunk
D*	Sanitized dataset
D′	Non-sensitive dataset
Sanitization time	Execution time = Total time taken by the algorithm to hide all the given sensitive information present in a dataset. In this paper, we have used both terms interchangeably
Sensitive itemset	Pattern of two or more items that exist together and may reveal some personal/confidential information that client do not want to disclose
Scalability	Capability of a system/ model/function/algorithm to cope and perform under an increased or expanding workload (data volume)
Outsourced techniques	All the privacy preservation techniques which ensure secure storage and computation of data at any third party like on a cloud
Data Hiding techniques	Techniques which distort or block sensitive patterns which can reveal any personal or confidential information before sharing data
Privacy level	It can be defined as the % of sensitive information masked before sharing data
Data quality	Ratio of knowledge content present in dataset after sanitization
Global index file	It maintains the sensitive itemset with corresponding supporting transaction Ids list and Delta value(Δ). It is maintained with the master node and shared with all other computing nodes in a cluster
Support index file	It maintains the list of 1-frequent itemset with global support i.e. with respect to the whole dataset. SIF can be accessed directly by any node in a cluster

Received: 2 September 2016 Accepted: 27 February 2017
Published online: 09 March 2017

References

1. Zhang X, Yang LT, Liu C, Chen J. A scalable two-phase top-down specialization approach for data anonymization using mapreduce on cloud. IEEE Trans Parallel Distrib Syst. 2014;25(2):363–73.
2. Liu F, Shu X, Yao D, Butt AR. Privacy-preserving scanning of big content for sensitive data exposure with MapReduce. In: Proceedings of the 5th ACM conference on data and application Security and Privacy. New York: ACM; 2015. p. 195–6.
3. Yi X, Rao FY, Bertino E, Bouguettaya A. Privacy-preserving association rule mining in cloud computing. In: Proceedings of the 10th ACM symposium on information, computer and communications security. New York: ACM; 2015. p. 439–50.
4. Huang C, Lu R. EFPA: efficient and flexible privacy-preserving mining of association rule in cloud. In 2015 IEEE/CIC international conference on communications in China (ICCC). New York: IEEE; 2015. p. 1–6.
5. Li J, Li J, Chen X, Liu Z, Jia C. Privacy-preserving data utilization in hybrid clouds. Fut Gener Comput Syst. 2014;30:98–106.
6. Wei L, Zhu H, Cao Z, Dong X, Jia W, Chen Y, Vasilakos AV. Security and privacy for storage and computation in cloud computing. Inf Sci. 2014;258:371–86.
7. Yan Z, Ding W, Niemi V, Vasilakos AV. Two schemes of privacy-preserving trust evaluation. Fut Gener Comput Syst. 2016;62:175–89.
8. Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. Commun ACM. 2008;51:107–13.
9. Oliveira SR, Zaiane OR. Privacy preserving frequent itemset mining. In: Proceedings of the IEEE international conference on privacy, security and data mining. Australian Computer Society, Inc.., 2002. Vol. 14, p. 43–54.

10. Oliveira SR, Zaïane OR. Protecting sensitive knowledge by data sanitization. New York: ICDM; 2003. Vol. 3, pp. 613–16.
11. Atallah M, Bertino E, Elmagarmid A, Ibrahim M, Verykios V. Disclosure limitation of sensitive rules. In: 1999 Workshop on knowledge and data engineering exchange, (KDEX'99) Proceedings. 1999. p. 45–52.
12. Amiri A. Dare to share: protecting sensitive knowledge with data sanitization. Decis Support Syst. 2007;43(1):181–91.
13. Zhang X, Liu C, Nepal S, Yang C, Chen J. Privacy preservation over big data in cloud systems. In: Security, Privacy and Trust in Cloud Systems. Berlin: Springer; 2014. p. 239–57.
14. Dasseni E, Verykios VS, Elmagarmid AK, Bertino E. Hiding association rules by using confidence and support. In: International workshop on information hiding. Berlin: Springer; 2001. p. 369–383.
15. Verykios VS, Elmagarmid AK, Bertino E, Saygin Y, Dasseni E. Association rule hiding. IEEE Trans Knowl Data Eng. 2004;16(4):434–47.
16. Bhandarkar M. MapReduce programming with apache Hadoop. In: IEEE international symposium on parallel and distributed processing (IPDPS), 2010. New York: IEEE; 2010.
17. Zhang Y, Cao T, Li S, Tian X, Yuan L, Jia H, Vasilakos AV. Parallel processing systems for big data: a survey. Proc IEEE. 2016;104(11):2114.
18. Fung BC, Wang K, Yu PS. Top-down specialization for information and privacy preservation. In: 21st international conference on data engineering (ICDE'05). New York: IEEE; 2005.
19. Sharma S, Toshniwal D. Parallelization of association rule mining: survey. In: International conference on computing, communication and security (ICCCS). New York: IEEE; 2015. p. 1–6.
20. The transactional data provided for acquire valued shoppers challenge. https://www.kaggle.com/c/acquire-valued-shoppers-challenge/data. Accessed 7 Oct 2016.