Predicate-Tree Based Pretty Good Privacy of Data*

William Perrizo and Arjun G. Roy

Department of Computer Science and Operations Research
North Dakota State University
Fargo, ND 58102, USA
{william.perrizo,arjun.roy}@ndsu.edu

Abstract. Growth of Internet has led to exponential rise in data communication over the World Wide Web. Several applications and entities such as online banking transactions, stock trading, e-commerce Web sites, etc. are at a constant risk of eavesdropping and hacking. Hence, security of data is of prime concern. Recently, vertical data have gained lot of focus because of their significant performance benefits over horizontal data in various data mining applications. In our current work, we propose a Predicate-Tree based solution for protection of data. Predicate-Trees or pTrees are compressed, data-mining-ready, vertical data structures and have been used in a plethora of data-mining research areas such as spatial association rule mining, text clustering, closed k-nearest neighbor classification, etc. We show how for data mining purposes, the scrambled pTrees would be unrevealing of the raw data to anyone except for the authorized person issuing a data mining request. In addition, we propose several techniques which come along as a benefit of using vertical pTrees. To the best of our knowledge, our approach is novel and provides sufficient speed and protection level for an effective data security.

Keywords: Predicate Trees, Data Mining, Data Security.

1 Introduction

Data communication over the Internet is at an all time high. Online banking transactions, realtime stock trading, ecommerce Web sites etc. rely completely on the security of World Wide Web and are at a constant risk of eavesdropping and hacking. Various data mining and analytic tools have come into existence which extract knowledge or meaningful information from massive amount of data stored locally or at a distant location. In this paper, we propose a Predicate-Tree or P-Tree based security solution for protection of data. P-Trees have been used in wide variety of data mining application. Our attempt in this paper is to propose a way of keeping the data secure as well as reap the benefits of using P-Trees. The next section gives a brief introduction on P-Trees followed by a section on PGP-D or Pretty Good Privacy of Data.

* We acknowledge partial financial support for this research from a Department of Energy Award (award # DE-FG52-08NA28921).

B. De Decker and D.W. Chadwick (Eds.): CMS 2012, LNCS 7394, pp. 192–194, 2012.

2 P-Tree

P-Trees are data-mining-ready, compressed and lossless data structures. The simplest form among them are the Peano-Trees which are bitwise trees comprising of only 0s and 1s. They can be 1, 2 or n-dimensional depending on the application. For e.g. finding the occurrence of a tuple, say (7,0,1,4) in a 4-attribute relational table can be efficiently computed by 1-dimensional P-Trees. For spatial images, 2-dimensional P-Trees are often used. P-Trees have been used in a wide variety of research areas as well including text mining [3], DNA Microarray data analysis [5], association rule mining [1], etc.

Construction of P-Trees is as follows: Let us consider a dataset X with d attributes represented as X = (A_1, A_2 ... A_d) and the binary representation of any k^{th} attribute, A_k, be represented as $b_{k,m-1}b_{k,m-2}b_{k,m-3}...b_{k,0}$. Here, m is the number of bits required to represent values in A_k. For e.g., 12 can be represented by 1100, so $m = 4$. Each of the attributes is decomposed into bit files, i.e. one file for each bit position. The P-Tree is simply constructed by taking bit files (one at a time) and recursively partitioning them into halves and sub-halves until each sub-half is absolutely pure i.e. entirely 1-bits or 0-bits.

Consider the following 1-attributed data containing values 2, 3, 2, 2, 5, 2, 7, 7. This is converted to binary resulting in three vertical strips of data (since each of the values can be represented by a 3-bit binary number). P_0, P_1 and P_2 are the three P-Trees generated for the above data. The construction process of each P-Tree is independant of other and thus can be parallelized over multicore processors.

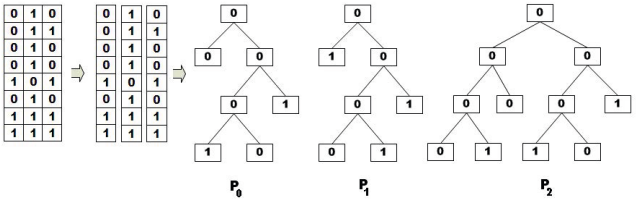

Fig. 1. P-Tree Construction

The most frequently used operations in P-Tree are the AND, OR and NOT logic operations. These operations are computed on a level by level basis starting from the root node. There are certain rules associated with these P-Tree operations. For example, an AND operation between a pure-0 node and any subtree results in a pure-0 node, an OR operation between a pure-1 node and any subtree results in the subtree itself, etc. Here, a P-Tree is said to be pure-1 if its subtree consists of only 1s and pure-0 otherwise. For more information on P-Tree structure, construction and operations, please refer to [4].

3 PGP-D

In the previous section, we described how data can be stored in the form of P-Trees which can readily be used for data mining. In this section, we propose how

the P-Trees can be secured from an attack. PGP-D is a mechanism in which we scramble P-Tree information(location information) in such a way that data can still be processed just as fast. For data mining purposes, the scrambled P-Trees would be unrevealing of the raw data to anyone, but a person qualified to issue data-mining requests (classification/ARM/clustering).

To retrieve P-Tree information, we require a) *ordering* - the mapping of the bit position to the table row b) *predicate* - table column id and bit slice or bitmap and c) *location*. The key of the data is an array of two tuples storing the *location* and the *pad*. A typical key could be something like [{5, 54}, {7, 539}, {87, 3}, {209, 126}, {25, 896}, {888, 23}, ...]. We make all the P-Trees of the same length and *pad* it in the front of each so that statistics can't reveal the start position. We also scramble the *location* of the P-Trees. For basic P-Trees, key K would reveal the offset and the pre-pad. For example, in the above key, the first P-Tree is found at offset 5, i.e. it has been shuffled forward 5 P-Tree slots from the slot initially assigned and that the 54 bits are *pad* bits.

Since P-Trees are data-mining-ready data structures, we are never in favor of making them go through the expensive process of encryption and decryption. Instead we focus on securing the key. Also, more the number of P-Trees, better is the protection. For a database with 5000 tables with 50 columns each and each column being represented by 32-bits, we would have 8 millions P-Trees. In distributed database scenario where we have multiple sites, it would make sense to fully replicate thus allowing all the retrieval as local. A condition could arise where the hacker extracts the first bit of every P-Tree (i.e. the 8,000,000,000 bits) that is the first horizontal record. He/She could shuffle those bits until something meaningful appears or starts to appear. From all the meaningful shuffles, he/she might be able to break the key code (e.g. look next as 2nd bit, then 3rd, etc.). To get around this possibility, we store the entire database as a massive "Big Bit String" and have it as a part of our key, the start offset of each P-Tree (which would be shuffled randomly).

References

1. Ding, Q., Ding, Q., Perrizo, W.: PARM - An Efficient Algorithm to Mine Association Rules from Spatial Data. IEEE Transactions on Systems, Man, and Cybernetics, Part B 38(6), 1513–1524 (2008)
2. Khan, M., Ding, Q., Perrizo, W.: k-nearest Neighbor Classification on Spatial Data Streams Using P-trees. In: Chen, M.-S., Yu, P.S., Liu, B. (eds.) PAKDD 2002. LNCS (LNAI), vol. 2336, pp. 517–528. Springer, Heidelberg (2002)
3. Rahal, I., Perrizo, W.: An optimized approach for KNN text categorization using P-trees. In: ACM Symposium on Applied Computing, pp. 613–617 (2004)
4. Perrizo, W.: Predicate Count Tree Technology. Technical Report NDSU-CSOR-TR-01-1 (2001)
5. Wang, Y., Lu, T., Perrizo, W.: A Novel Combinatorial Score for Feature Selection with P-Tree in DNA Microarray Data Analysis. In: 19th International Conference on Software Engineering and Data Engineering, pp. 295–300 (2010)