Analysis of encryption algorithms to basic operations on outsourced B+ trees for data privacy

Conference Paper · April 2011

DOI: 10.1109/ICECTECH.2011.5942124

2 authors:

Sunil B. Mane
College of Engineering, Pune
14 PUBLICATIONS **51** CITATIONS
SEE PROFILE

Pradeep Sinha
69 PUBLICATIONS **484** CITATIONS
SEE PROFILE

Analysis of Encryption Algorithms to Basic Operations on Outsourced B^+ Trees for Data Privacy

Mr. Sunil B. Mane
Asst. Professor, Computer Engineering & Information Technology Department
College of Engineering, Pune,
M.S., India
E-mail: sunilbmane.comp@coep.ac.in

Dr. Pradeep K. Sinha
Director (High Performance Computing) Center for Development & Advance Computing,
Pune, M.S., India
E-mail: psinha@cdac.in

***Abstract*—Shared data systems have grown in numbers, many of these are critical in nature. As a recent trend, there has been growing interest in outsourcing database services in both the commercial world and the research community. In the outsourced database service (ODBS) model, clients rely upon the service provider, which include hardware, software and manpower, for the storage, maintenance, and retrieval of their data. Hence, it tends to variety of security challenges involved in it.**

The outsourced B+ index trees and data are encrypted and stored at some un-trusted server. Here, we discuss implementation of data security issues on outsourced B+ index trees and ensure security in the execution of basic operations like search and updates (insert, delete) on the un-trusted server. Our proposed work aims at presenting the results after implementing basic operations on outsourced B+ index trees with the help of various encryption algorithms to ensure data privacy.

Keywords- Outsourced Databases; Encryption Algorithms; Data Privacy;

I. INTRODUCTION

Search trees are basically multilevel indexes which can be used to cluster many varieties of data ranging from one-dimensional to multi-dimensional spatial data sets as given in [1]. Proper index trees are often built for convenient access to data in the databases. To retrieve an indexed data item, a client needs to traverse the index tree to find the location of data. In order to hide the query and data present in each node of B^+ index tree, the traversal path needs protection mentioned in [2,5]. In the proposed technique, clients learn how to privately traverse a remotely stored tree structure to locate and retrieve data. The tree structure and the traversal path are hidden from the server. Basic operations of search trees include search (to provide solutions for different types of queries) and updates (modification, insert, delete). In practical scenario, update operations are complex to implement, especially in case of *dynamic databases*, because they heavily degrade the performance in order to create the structure of the resulting search trees.

II. TRAVERSAL OF TREE STRUCTURE

On outsourcing private data with the help of search trees, the tree structure and data should all be confidential. Thus it is preferred to encrypt each tree node as a whole because protecting a tree-based index by encrypting each of its fields would leak the ordering relationship between the index values to the server as given in [4]. Each node is identified by a unique node identifier (NID). In this paper, we have discussed the challenges to privately retrieve hidden tree structured data, especially how to let a client traverse a tree structure to find the desired data node as shown in figure1, with minimizing the leakage of the tree structure and the target data. So we have worked on Lin and Candan's approach [7, 9] for B+ index trees as mentioned below.

- *Limitations in storage space usage and Maintenance of empty node lists: In* each node swapping operation, the target node is swapped with an empty node.

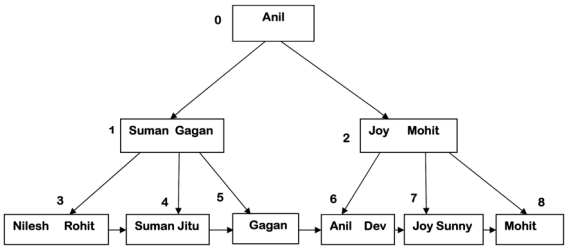

Figure 1. An Example of B + Trees

- *Insert and delete operations on outsourced search trees:* Our proposed design supports to facilitate these basic operations.
- *Maintaining the tree structure integrity:* It is observe that the solution to the tree structure integrity maintenance can be applicable to insert and delete operations in dynamic outsourced databases even when the node is split and deletion of over-full and under-full nodes occurs.
- *No need of more empty nodes for a redundancy set:* The node swapping technique using only one empty node for each redundancy set is suitable for cases in which an over-full node is split into two new nodes.
- *Indexed data modification:* The object location in the search tree is changed whenever attributes or values of a data object are modified.

978-1-4244-8679-3/11/$26.00 ©2011 IEEE

- *Managing empty node lists:* Empty nodes are stored in hidden linked lists. To help clients find out the empty nodes, two other types of pointers are also stored in the Special node: HEAD and TAILS point to the heads and tails of empty node lists, respectively.

So, for all the above provided issues and limitations, we had worked on the given solution [7] based on the practical algorithms for all types of outsourced B+ index trees.

III. Operations to provide data security on Outsourced Search Trees

In this section, we will present algorithms that have been implemented to ensure data security for outsourced search trees.

A. Managing Empty Nodes

Objective of our work is firstly to keep information about empty nodes up to-date after each node is swapped. Secondly to provide an efficient use for the database storage space at the server for which a hidden linked list is used to keep empty nodes and information about these lists is stored in the special node.

B. Steps in Search Operation

This algorithm is same as that of insert (3.3), the only difference is that we don't have to split the node and perform any insert operation.

- Get Name of the table and Data to be inserted from user
- At server, generate empty node list. Empty nodes are the nodes having no data or any data that can be identified as null data. Each node of empty node list is called ENODE.
- Get root address in any random ENODE. The address of this ENODE is known to client only. This root is the root of B+ tree.
- At server side begin fetching nodes.
- Generate redundancy set for node.
- Attach three empty nodes to the redundancy set.
- Send this array to the client
- At client side, decrypt the node and check whether the required node is present in the array or not. For B+ trees all the operations are performed on leaf. So we have to go to the leaf node.
- Continue steps 5 to 9 till client gets target leaf node.
- At client side when target node is found, decrypt data perform operations and then re encrypt the node. Swap this node with empty node so that the address of target node gets changed.
- Complete the operation.

C. Insert Operation

To insert a new data object into a search tree, we have to look first for a leaf node that is *most suitable* to keep this new object. After a leaf is decided and if it still has at least one free entry, then the new object will be inserted into that leaf. Otherwise, the leaf is a full node and needs to be split into two leaf nodes, and the new object is inserted into one of them[6,8].

In this case, corresponding updates on the parent node must be performed to reflect the split, and such updates may propagate up the tree if the parent node is also already full. In the worst case, the splits propagate up to the root, and if the root is also full, it must be split and a new root is created, so the tree height increases.

We have to lock nodes along the traversal path on the tree in case of outsourced search trees. If the nodes are not locked, later necessary updates are impossible because the NIDs can be changed by other oblivious operations.

Steps in Insert Operation:

- Get Name of the table and Data to be inserted from user
- At server, generate empty node list. Empty nodes are the nodes having no data or any data that can be identified as null data. Each node of empty node list is called ENODE.
- Get root address in any random ENODE. The address of this ENODE is known to client only. This root is the root of B+ tree.
- At server side begin fetching nodes.
- Generate redundancy set for node.
- Attach three empty nodes to the redundancy set.
- Send this array to the client
- At client side, decrypt the node and check whether the required node is present in the array or not. For B+ trees all the operations are performed on leaf. So we have to go to the leaf node.
- If the node is internal node, split the node and copy parent, left and right child to three empty nodes. Return the node to server. Splitting is performed at client side so that server is unaware of parent and child node relationship.
- Continue steps 5 to 9 till client gets target leaf node.
- At client side when target node is found, insert data to the node and then re encrypt the node. Swap this node with empty node so that the address of target node gets changed.
- Complete the operation.

D. Delete Operation

When an object is deleted from a leaf, more issues are arisen if that leaf becomes under-full. With this policy, all objects in an under-full leaf are reinserted after this leaf is removed from the tree.

Steps in Delete Operation:

- Get Name of the table and Data to be inserted from user
- At server, generate empty node list. Empty nodes are the nodes having no data or any data that can be identified as null data. Each node of empty node list is called ENODE.
- Get root address in any random ENODE. The address of this ENODE is known to client only. This root is the root of B+ tree.
- At server side begin fetching nodes.
- Generate redundancy set for node.
- Attach three empty nodes to the redundancy set.
- Send this array to the client
- At client side, decrypt the node and check whether the required node is present in the array or not. For B+ trees all the operations are performed on leaf. So we have to go to the leaf node.
- Continue step 5 to 8 till client gets target leaf node.
- If the node is having fewer keys than fill factor which is 50% for b+ trees then, go to parent node and again send this node to client.
- Client performs necessary adjustments in the tree.
- Client returns the parent node and then the operation continues.
- The delete operation ends with client deleting the required key from target node. This node is then re-encrypted and swapped with empty node in the array.
- The array is sent back to the server.
- Complete the operation.

E. Update Operation

Update operation can be considered as delete followed by insert operation.

Steps in Update Operation:

- Get Name of the table and Data to be inserted from user
- At server, generate empty node list. Empty nodes are the nodes having no data or any data that can be identified as null data. Each node of empty node list is called ENODE.
- Get root address in any random ENODE. The address of this ENODE is known to client only. This root is the root of B+ tree.
- At server side begin fetching nodes.
- Generate redundancy set for node.
- Attach three empty nodes to the redundancy set.
- Send this array to the client
- At client side, decrypt the node and check whether the required node is present in the array or not. For B+ trees all the operations are performed on leaf. So we have to go to the leaf node.
- If the node is internal node, split the node and copy parent, left and right child to three empty nodes. Return the node to server. Splitting is performed at client side so that server is unaware of parent and child node relationship.
- Continue steps 5 to 9 till client gets target leaf node.
- At client side when target node is found, insert data to the node and then re encrypt the node. Swap this node with empty node so that the address of target node gets changed.
- If the node is having fewer keys than fill factor which is 50% for b+ trees then, go to parent node and again send this node to client.
- Client performs necessary adjustments in the tree.
- Client returns the parent node and then the operation continues.
- The Update operation ends with client deleting the required key from target node. This node is then re-encrypted and swapped with empty node in the array.
- The array is sent back to the server.
- Complete the operation.

IV. Design and Implementation

Following Figure 2 shows the proposed system design and implementation details. Mainly it includes three components as follows:

- *Client:* Client is just a browser from where the request is generated and the results are displayed.
- *Application Server:* It is responsible to handle the HTTP request/reply. It also acts as communication interface between the client and Database Server.
- *Database Server:* This component stores the actual database. It communicates with the application server with servelets. Mysql is used as database engine.

In the proposed system, a client sends request to application server (Apache Tomcat) through web browser and accesses it with the help of HTTP request/reply. An application server connects to client servlets which handle all the request for encryption and decryption of data, further the request is passed to the database server which connects to remote database that has inbuilt business logic which keeps all the information regarding data like in what format it is to be stored, where it is to be stored, etc. The database server contains servlets that does the processing of all operations and fetches the required data from database. After processing the results are sent back to the application server which does the necessary changes according to the

need of client and sends it back to client side through web browser.

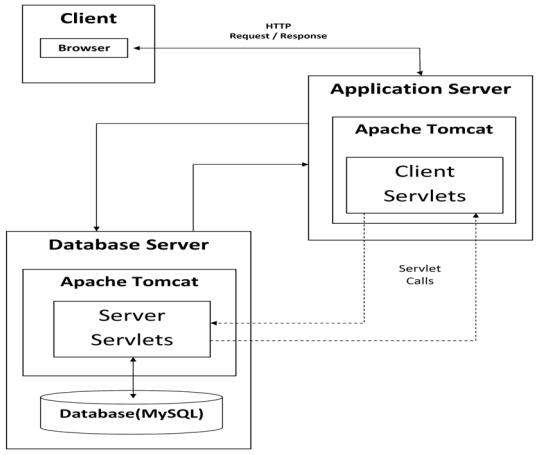

Figure 2. Client Server Architecture

V. EXPERIMENTATION AND RESULTS

As per the algorithm given in [7], we have designed the architecture as shown in Figure 2 and implemented all above algorithms using java language. We have executed the algorithms on different databases with different data types namely character, float, integer and a combination of all. We stored all above types of databases by using B+ index Tree approach. We have implemented the encryptions techniques particularly Advanced Encryption System (AES), Data encryption system (DES) and Blowfish. We have calculated the running time for each operation with respect to different databases with different data types. The results we calculated have been presented in the form of graphs as shown in Figure 3, Figure 4, and Figure 5.

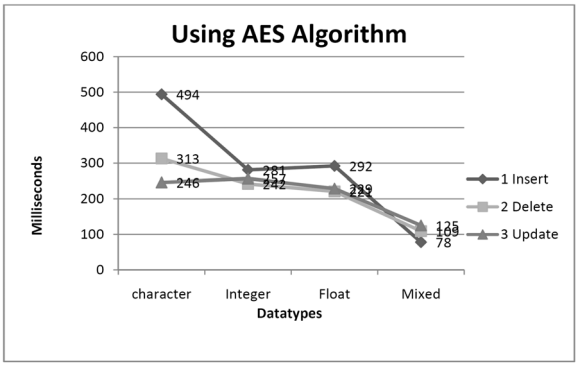

Figure 3. Results for AES Algorithm

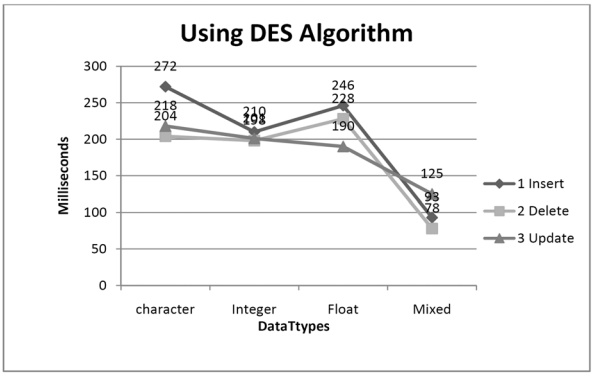

Figure 4. Results for DES Algorithm

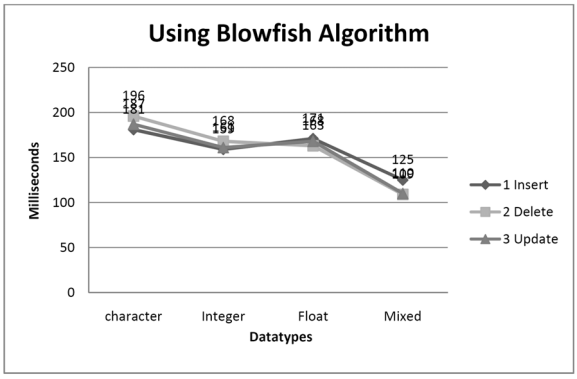

Figure 5. Results for Blowfish Algorithm

VI. CONCLUSIONS

Hence, we have implemented solutions to the problem of preserving privacy for basic operations on outsourced B+ trees overcoming the limitations of [7]. The results we see as in Figure 3, Figure 4, and Figure 5. It infers that insert operation takes larger amount of time followed by delete operation then by update operations. Based upon the results by using different encryption algorithms AES algorithm takes more time compared to DES and which takes more time compared to the Blowfish algorithm for the implementing B+ index trees on outsourced database for data privacy.

REFERENCES

[1] R. Bayer. "The Universal B-Tree for Multidimensional Indexing: General Concepts." WWCA'97, Japan, 1997.

[2] L. Bouganim, P. Pucheral. Chip-Secured Data Access: "Confidential Data on Untrusted Servers." VLDB 2002

[3] T.K. Dang. "Extreme Security Protocols for Outsourcing Database Services." The 6th iiWAS Conf., Jakarta, Indonesia, Sept. 2004, pp. 497-506

[4] E. Damiani, S.D.C. Vimercati, S. Jajodia, S. Paraboschi, P. Samarati. "Balancing Confidentiality and Efficiency in Untrusted Relational DBMSs". The 10th ACM Conf. on Computer and Comm. Security, USA, 2003

[5] H. Hacigümüs, B.R. Iyer, C. Li, S. Mehrotra. Executing “SQL over Encrypted Data in the Database-Service-Provider Model.” ACM SIGMOD Conf., 2002

[6] H. Hacigümüs, S. Mehrotra, B.R. Iyer. “Providing Database as a Service.” ICDE, USA, 2002

[7] Tran Khan Dang, “Privacy-Preserving Basic Operations on Outsourced Search Trees” (paper presented at the International Workshop on Privacy Data Management – PDM05, in conjunction with ICDE05, IEEE Computer Society, Tokyo, Japan, April 2005).

[8] Dawn Xiao dong Song, David Wagner, and Adrian Perrig “Practical Techniques for Searches on Encrypted Data” (paper presented at the IEEE Symposium on Security and Privacy, Berkeley, CA, USA, May 2000), 44-55.

[9] P. Lin, K.S. Candan. “Hiding Traversal of Tree Structured Data from Untrusted Data Stores.” The second International Workshop on Security in InformationSystems, Porto, Portugal, April 2004, pp. 314-323.