Query Optimization in Encrypted Database Systems

Hakan Hacıgümüş[1], Bala Iyer[2], and Sharad Mehrotra[3]

[1] IBM Almaden Research Center, USA
hakanh@acm.org
[2] IBM Silicon Valley Lab., USA
balaiyer@us.ibm.com
[3] University of California, Irvine, USA
sharad@ics.uci.edu

Abstract. To ensure the privacy of data in the relational databases, prior work has given techniques to support data encryption and execute SQL queries over the encrypted data. However, the problem of how to put these techniques together in an optimum manner was not addressed, which is equivalent to having an RDBMS without a query optimizer. This paper models and solves that optimization problem.

1 Introduction

There is an ongoing consolidation in IT industry that results in the application-service-provider (ASP) model. Organizations outsource some or all of their core IT operations (e.g., data centers) to specialized service providers over the Internet. Many users will be storing their data and processing their applications at the remote, potentially untrusted, computers. One of the primary concerns is that of data privacy – protecting data from those who do not need to know.

There are two kinds of threats to the privacy. Outsider threats from hackers and insider threats, perhaps, disgruntled employees. Encrypting the stored data [3] is a way to address the outsider threats. The data is only decrypted on the server before computation is applied and re-encrypted thereafter. The insider threats, however, are more difficult to protect against. For example, how would one protect the privacy of data from the database system administrator who probably has the superuser access privileges? If the client is on a secure environment then one way to solve the insider threat problem is to store all the data encrypted on the server and make it impossible to decrypt on the server. In this model we assume the computation against the data stored at the server is initiated by the client. Say it is possible to transform and split the computation into two parts. The server part is sent to the server to execute directly against the encrypted data giving encrypted results, which are shipped to the client, who decrypts and performs the end user portion of the computing. This scheme, presented [1], addresses the problem of insider threats. The problem of how to perform this scheme in an optimum manner was not addressed, which

L. Zhou, B.C. Ooi, and X. Meng (Eds.): DASFAA 2005, LNCS 3453, pp. 43–55, 2005.

is equivalent to having an RDBMS without a query optimizer. In this paper, we essentially address that problem. We will start our presentation with the description of the system model.

2 The System Model

In this paper, we follow the service-based database model. We specifically follow the system model described in [1]. In the model, the data is owned by the clients. The server exposes mechanisms for the clients to create tables, insert, update records and execute queries. The privacy challenge is to make it impossible for the server to correctly interpret the data.

The data originates from the client and the data is encrypted by the client before it is sent to the server for inclusion in a table. The data is always encrypted when it is stored on, or processed by the server. The authorized clients are given the needed encryption key(s). At no time is the encryption key given to the server, thus the data cannot be decrypted by the server. Queries against data-in-the-clear, originate from the client. The algorithms, based on client metadata, decompose the query into client and server queries. The server query is sent to the server to execute against encrypted data. The processing algorithms are designed such that the results of the original query are obtained if the client decrypts and further processes the answers of the server query using the decomposed client query.

Encrypted Data Storage Model: The encrypted storage model defines how the clients' data is stored at the server site in encrypted form. The storage model we use in this study substantially enhances the one presented in [1] and we presented essential parts of it elsewhere [4]. Hence we give the details of the storage model in Appendix A for the benefit of the reader.

3 Query Processing over Encrypted Data

Given a query Q, which is represented as an operator tree, our purpose is to define how the query can be securely evaluated in an encrypted database environment. We partition a given query tree into two parts: Q^S and Q^C, where Q^S executes at the server and Q^C at the client. Since decryption is not allowed at the server, Q^S executes over the encrypted data directly. The most obvious way to partition the query processing in this case is to store the encrypted tables at the server side and transfer them whenever they are needed for the query processing to the client. Then, the client could decrypt the tables and evaluate the rest of the query. Although this model would work, it pushes almost the whole query processing to the client and does not allow the client to exploit the resources available at the server site. In computing models we consider, the goal of the partitioning is to minimize the work done by Q^C since the clients may have limited storage and computational resources and they rely on the servers for the bulk of the computation.