HILL CLIMBING ALGORITHM FOR DATA DISTRIBUTION IN SECURE DATABASE SERVICES

Rahul Meshram*

S. J. Karale**

ABSTRACT

Recent trends of outsourcing databases to the third party database service providers have led to great interest in enabling secure database services. Previous approaches to enabling such a service have been based on data encryption, causing a large overhead in query processing. We propose an architecture which allows an organization to outsource its data management to two servers while preserving data privacy. Distributed architecture provides good privacy and fault tolerance to the client. The proposed framework integrates (1) the heuristic module which defines a set of heuristics to drive the fragmentation of object databases and incorporates them in a methodology that includes an analysis algorithm, horizontal and vertical class fragmentation algorithms. (2) The query at the client to queries for the servers is done by a bottom up state based algorithm. Finally the results at the servers are integrated to obtain the answer at the client.

Keywords: Data Confidentiality, Distributed database, Hill Climbing Algorithm

*Yeshwantrao Chavan College of Engineering, Nagpur.

** Assistant Professor, Yeshwantrao Chavan College of Engineering, Nagpur.

1. INTRODUCTION

Database service providers are becoming ubiquitous these days. These are companies which have the necessary hardware and software setup (data centers) for storage and retrieval of terabytes of data [4, 5, 6]. As a result of such service providers, parties wanting to store and manage their data may prefer to outsource data to these service providers. The parties who outsource their data will be referred to as clients hereafter. The service providers storing data will be referred to as servers. There is a growing concern regarding data privacy among clients. Often, client data has sensitive information which they want to prevent from being compromised. Examples of sensitive databases include a payroll database or a medical database. To capture the notions of privacy in a database, privacy constraints are specified by the client on the columns of the sensitive database. We apply heuristic search techniques based on Greedy Hill Climbing to come up with nearly optimal solutions.

A client using a database service needs to trust the service provider with potentially sensitive data, leaving the door open for damaging leaks of private information. Consequently, there has been much recent interest in a so-called Secure Database Service - a DBMS that provides reliable storage and efficient query execution, while not knowing the contents of the database [7]. Such a service also helps the service provider by limiting their liability in case of break-ins into their system - if the service providers do not know the contents of the database, neither will a hacker who breaks into the system.

The distribution design involves making decisions on the fragmentation and placement of data across the sites of a computer network. The first phase of the distribution design in a top-down approach is the fragmentation phase, which is the process of clustering in fragments the information accessed simultaneously by applications. The fragmentation phase is then followed by the allocation phase, which handles the physical storage of the generated fragments among the nodes of a computer network, and the replication of fragments. This work addresses the fragmentation phase of databases. We believe that, by outputting good fragmentation schemas with improved performance, data allocation and replication may then be carried out more efficiently, since the fragmentation schema will adequately reflect appropriate units of distribution according to the application access patterns, and thus may significantly reduce the search space of the allocation phase. To fragment a class, it is possible to use two basic techniques: horizontal fragmentation and vertical fragmentation. In object databases, horizontal fragmentation distributes class instances across the fragments. Thus, a horizontal fragment of a class contains a subset of the whole class extension. On the

other hand, vertical fragmentation (VF) breaks the class logical structure (its attributes and methods) and distributes them across the fragments. The horizontal fragmentation is usually subdivided in primary and derived horizontal fragmentation. Primary horizontal fragmentation (PHF) basically optimizes set operations (search over a class extension), firstly by reducing the amount of irrelevant data accessed and, secondly, by permitting applications to be executed concurrently, thus achieving a high degree of parallelism. On the other hand, derived horizontal fragmentation (DHF) can be viewed as an approach of clustering objects of distinct classes in the disk, therefore clearly addressing the relationships between classes and improving performance of applications with navigational access. It is also possible to apply both vertical and horizontal fragmentation techniques in a class simultaneously (which we call hybrid fragmentation) or to apply different fragmentation techniques in different classes in the database schema (which we call mixed fragmentation).

2. RELATED WORK

To capture the notions of privacy in a database, privacy constraints are specified by the client on the columns of the sensitive database. We use the notion of privacy constraints as described in [3, 8]. An example of a privacy constraint is (age, salary) which states that age and salary columns of a tuple must not be accessible together at the servers. The clients also have a set of queries also known as the workload that need to executed on a regular basis on their outsourced database. Most existing solutions for data privacy rely on encrypting data at the server, so that only the client can decrypt it (see for example [9, 10]). Unfortunately, it is hard to run general queries on encrypted data efficiently. If the server cannot execute parts of a query, it sends a fraction of the encrypted database back to the client for further filtering and processing, clearly an expensive proposition. There has been much recent interest in a so-called Secure Database Service - a DBMS that provides reliable storage and efficient query execution, while not knowing the contents of the database [7]. Such a service also helps the service provider by limiting their liability in case of break-ins into their system - if the service providers do not know the contents of the database, neither will a hacker who breaks into the system. Existing proposals for secure database services have typically been founded on encryption [11, 12, 13]. Data is encrypted on the (trusted) client side before being stored in the (untrusted) external database. Observe that there is always a trivial way to answer all database queries: the client can fetch the entire database from the server, decrypt it, and execute the query on this decrypted database. Of course, such an approach is far too expensive to be practical. Instead, the hope is that queries can be transformed by the client to

execute directly on the encrypted data; the results of such transformed queries could be post-processed by the client to obtain the final results.

3. GENERAL ARCHITECTURE

The general architecture of a distributed secure database service, as illustrated in Figure 1, consists of a trusted client as well as two or more servers that provide a database service. The servers provide reliable content storage and data management but are not trusted by the client to preserve content privacy.

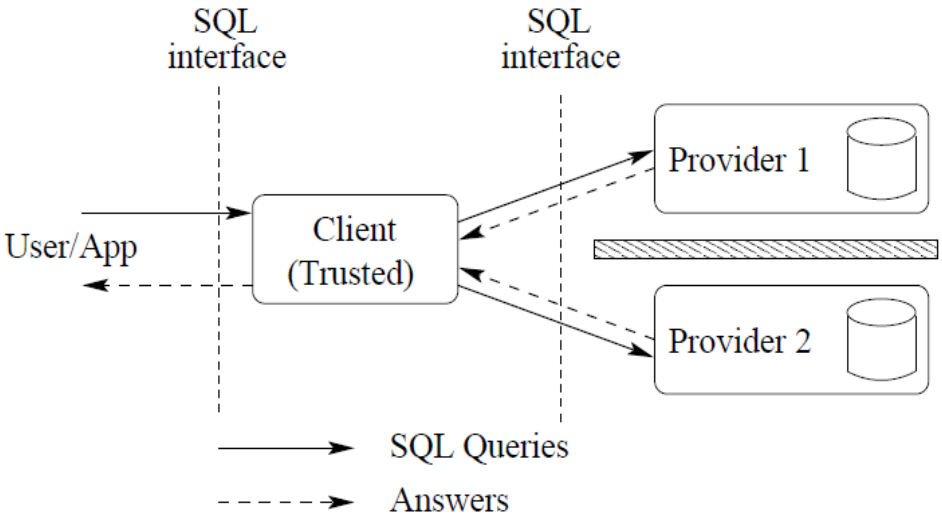

Figure 1: System Architecture

The client wants to out-source the (high) costs of managing permanent storage to the service providers; hence, we assume that the client does not store any persistent data. However, the client has access to cheap hardware - providing processing power as well as temporary storage - which is used to provide three pieces of functionality:

1. Offer a DBMS Front-End the client exports a standard DBMS front-end to client-side applications, supporting standard SQL APIs. 2. Reformulate and Optimize Queries The queries received by the client need to be translated into appropriate SQL sub-queries to be sent to the servers; such translation may involve limited forms of query-optimization logic, as we discuss later in the paper. 3. Post-process Query Results the sub-queries are sent to the servers (using a standard SQL API), and the results are gathered and post-processed before being returned in a suitable form to the client-side application. We note that all three pieces of functionality are fairly cheap, at least if the amount of post-processing required for queries

is limited, and can be performed using inexpensive hardware, without the need for expensive data management infrastructure or personnel. Security Model As mentioned earlier, the client does not trust either server to preserve data privacy. Each server is honest, but potentially curious: the server may actively monitor all the data that it stores, as well as the queries it receives from the client, in the hope of breaching privacy; it does not, however, act maliciously by providing erroneous service to the client or by altering the stored data. The client maintains separate, permanent channels of communication to each server. We do not require communication to be encrypted; however, we assume that no eavesdropper is capable of listening in on both communication channels. The two servers are assumed to be unable to communicate directly with each other (depicted by the \wall" between them in Figure 1) and in fact, need not even be aware of each other's existence. Note that the client side is assumed to be completely trusted and secure. There would not be much point in developing a secure database service if a hacker can simply penetrate the client side and transparently access the database. Preventing client-side breaches is a traditional security problem unrelated to privacy-preserving data storage, and we do not concern ourselves with this problem here.

4. PROPOSED SCHEAM

We develop a heuristic search strategy that finds good, although not optimal, solutions. Figure 2 illustrates our approach to finding good partitions. At the core is a hill climbing module that tries to improve on an existing partition. This module starts with an initial, simple partition that satisfies the privacy constraints, and then makes local changes to the partition that still satisfy the constraints. To decide if a partition is better than the current one, the hill climbing module must compare their costs. For this comparison, it uses two modules: (1) the translation engine that given a workload query and a partition, determines the execution plan (what sub-queries are sent to the servers); and (2) the query cost estimator that estimates the cost of a given plan.

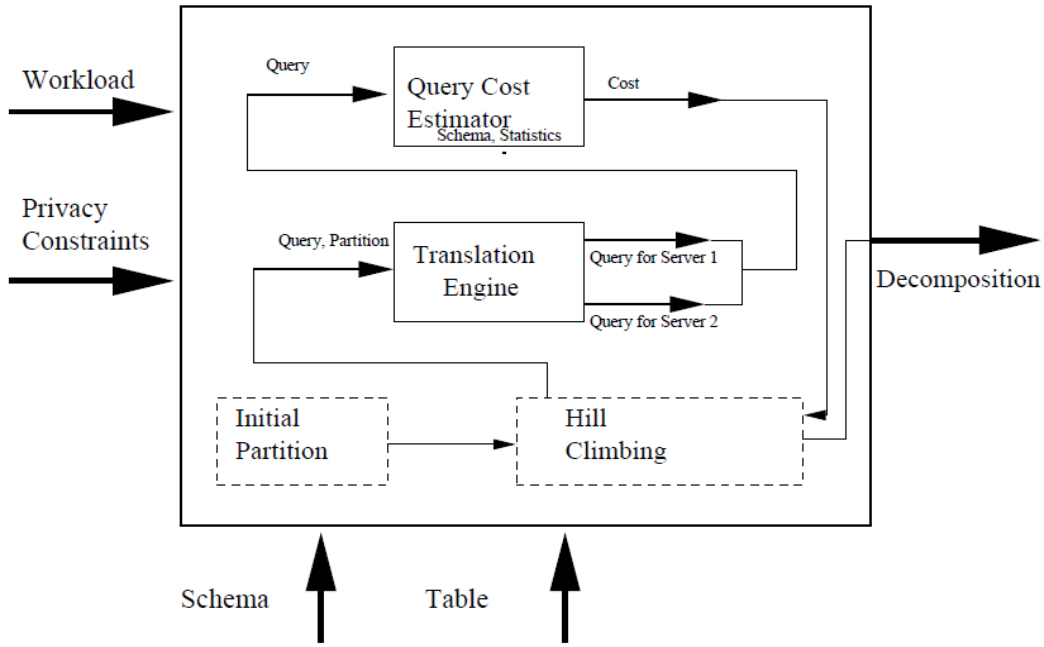

Figure 2. Components of the Partitioning Algorithm

1 Query cost estimator

In order to perform cost estimation, we collect and maintain statistics of the data. For a given relation R, we maintain the following information.

T (R): Number of tuples in R

S(R, a): Size in bytes of attribute a in R

V (a): Number of distinct values of a in R

Let F be a boolean predicate which is given by the grammar

We use ρ(F) to denote the selectivity of the formula F. ρ(F) is computed recursively using the semantics.

The attributes in the SELECT clause of the query decide the size in bytes of each result tuple. Query cost, QC(q) represents the size estimation for query q and SL(q) is the set of attributes in the SELECT clause for q. The cost estimate for a partition is computed as the sum of the cost estimate of the two queries.

2 Translations and Execution Engine

The translation engine is the system component which generates SQL queries for the decomposition D(R) of R, given a SQL query on R. The partitioned queries generated by the engine can now be fed to the query estimator discussed in the previous section to obtain cost estimates for each query. The type of plan used is an important factor which decides the form of the resulting queries. For the purposes of this paper, we generate queries for **centralized plans.**

This problem of deciding which server to use, to access data is better known are data localization in distributed databases theory as discussed in [14]. Replication and encryption add more complexity to the localization process. For example, if an attribute is available at both servers, one decision to make is which copy must be accessed. Range queries on encrypted attributes will require the entire column to be transmitted to the client for decryption before determining the results of the query. Decisions like which copy to access cannot be determined locally and individually for each condition clause.

We propose a technique which computes the where clauses in the decomposed queries in two steps. We define two types of state values, W and S each of which provide information as to which servers to access for the query execution. We process the WHERE clause to get W and then process the SELECT clause to get S. In the final step, we use both these values and the corresponding select and condition list to determine the decomposed queries. We use the schema R and decomposition D(R) defined in section 2 Most of the steps that follow are part of query localization which is to decide which part of the query is processed by which server.

5. PARTITIONING ALGORITHM

Hill-climbing is a heuristic in which one searches for an optimum combination of a set of variables by varying each variable one at a time as long as the objective value increases. The algorithm terminates when no local step decreases the cost. The algorithm converges to a local minima. An initial fragmentation of the database is considered which satisfies all the privacy constraints.

Initial Guess: The initial state is obtained using the weighted set cover. Refer [15] for details of the algorithm. Algorithm for Weighted Set Cover: - Assign a weight to each attribute based on the number of privacy constraints it occurs in. - Encrypt attributes one at a time starting with the one which has the highest weight till all the privacy constraints are satisfied.

Hill Climbing Step: Then, all single step operations are tried out (1) Decrypting an encrypted column and placing it at Server 1. (2) Decrypting an encrypted column and placing it at Server 2. (3) Decrypting an encrypted column and placing it at both servers (4) Encrypting an decrypted column and placing it at both servers. From these steps, the one which satisfies privacy constraints and results in minimum network traffic is considered as the new fragmentation and the process repeats. The iterations are performed as long as we get decomposition at each step which improves over the existing decomposition using the cost metric discussed before.

In order to compare the results produced by our hill climbing strategy with the optimal solution, we also implemented a brute force algorithm. This algorithm considers all possible partitions that satisfy the privacy constraints and selects the one with minimal cost.

Note that for a relation with n columns there are 4n possible fragmentations possible and very few of them will satisfy all the privacy constraints (The "4" arises because there are 4 choices for each attribute: store decrypted at server 1 or 2 or both, or store encrypted at both servers.)

6. CONCLUSION AND FUTURE WORKS

We have introduced a new distributed architecture for enabling privacy-preserving outsourced storage of data. We demonstrated different techniques that could be used to decompose data, and explained how queries may be optimized and executed in this distributed system. We introduced a definition of privacy based on hiding sets of attribute values, demonstrated how our decomposition techniques help in achieving privacy, and considered the problem of identifying the best privacy-preserving decomposition. Given the increasing instances of database outsourcing, as well as the increasing prominence of privacy concerns as well as regulations, we expect that our architecture will prove useful in ensuring compliance with laws and in reducing the risk of privacy breaches.

A key element of future work is to test the viability of our architecture through a real-world case study. Other future work includes identifying improved algorithms for decomposition, expanding the scope of techniques available for decomposition, e.g., supporting replication, and incorporation of these techniques into the query optimization framework.

REFERENCES

[1] Vignesh Ganapathy, Dilys Thomas, Tomas Feder, Hector Garcia-Molina, Rajeev Motwani. Distributing Data for Secure Database Services, 2011 ACM.

[2] G. Aggarwal, T. Feder, K. Kenthapadi, R. Motwani, R. Panigrahy, D. Thomas, and A. Zhu. Anonymizing tables. In *Proceedings of the International Conference on Database Theory*, pages 246–258, 2005.

[3] G. Aggarwal, M. Bawa, P. Ganesan, H. Garcia-Molina, K. Kenthapadi, R. Motwani, U. Srivastava, D. Thomas, and Y. Xu. Two can keep a secret: A distributed architecture for secure database services. In *Conference on Innovative Data Systems Research*, 2005.

[4] Amazon. Amazon elastic compute cloud. Available from URL:http://www.amazon.com/b/ref=scfel2/?node=201590011&no= 3435361.

[5] Google. Google apps for your domain. Available from URL:http://www.google.com/a/.

[6] Salesforce. Salesforce on-demand customer relationship management. Available from URL:http://www.salesforce.com/.

[7] Murat Kantarcioglu and Chris Clifton. Security issues in querying encrypted data. Technical Report TR-04-013, Purdue University, 2004.

[8] A. Motro and F. Parisi-Presicce. Blind custodians: A database service architecture that supports privacy without encryption. In *International Federation for Information Processing*, 2005.

[9] H. Hacigumus, B. Iyer, C. Li, and S. Mehrotra. Executing SQL over encrypted data in the database-service-provider model. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2002.

[10] H. Hacigumus, B. Iyer, and S. Mehrotra. Providing database as a service. In *Proceedings of the International Conference on Data Engineering*, 2002.

[11] H. Hacigumus, B. Iyer, C. Li, and S. Mehrotra. Executing sql over encrypted data in the database-service-provider model. In Proc. SIGMOD, 2002.

[12] H. Hacigumus, B. Iyer, and S. Mehrotra. E_cient execution of aggregation queries over encrypted relational databases. In Proc. DASFAA, 2004.

[13] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order-preserving encryption for numeric data. In Proc. SIGMOD, 2004.

[14] M. T. Ozsu and P. Valduriez. Principles of Distributed Database Systems. Prentice Hall, 2nd edition, 1999.

[15] V. Vazirani. Approximation Algorithms. Springer, 2004.