Distributing Data for Secure Database Services

Vignesh Ganapathy
Stanford University
vignesh@cs.stanford.edu

Dilys Thomas
TRDDC, Pune India
dilys@cs.stanford.edu

Tomas Feder
Stanford University
tomas@theory.stanford.edu

Hector Garcia-Molina
Stanford University
hector@cs.stanford.edu

Rajeev Motwani
Stanford University
rajeev@cs.stanford.edu

Abstract

The advent of database services has resulted in privacy concerns on the part of the client storing data with third party database service providers. Previous approaches to enabling such a service have been based on data encryption, causing a large overhead in query processing. A distributed architecture for secure database services is proposed as a solution to this problem where data is stored at multiple servers. The distributed architecture provides both privacy as well as fault tolerance to the client. In this paper we provide algorithms for (1) distributing data: our results include hardness of approximation results and hence a heuristic greedy algorithm for the distribution problem (2) partitioning 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. We provide an experimental validation and performance study of our algorithms.

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 [8, 15, 25]. 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*.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and or a fee.
PAIS'11, 25-MAR-2011, Uppsala, Sweden.
Copyright 2011 ACM 978-1-4503-0611-9/11/03...$10.00.

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 use the notion of privacy constraints as described in [3, 23]. 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 [16, 17]). 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.

Instead, Reference [3] suggests using two (multiple) service providers in order to store the data. The advantage of using two servers is that the columns can be split across the two servers to satisfy privacy constraints without encrypting the split columns. Thus, in order to satisfy privacy constraints, columns can either be split across servers or stored encrypted. Thus the goal of any decomposition algorithm is to partition the database to satisfy the following.

- None of the privacy constraints should be violated.
- For a given workload, minimum number of bytes should be transferred between the servers and the client.

We explain both of the above points in detail in the next section. The problem of finding the optimal partition structure for a given set of privacy constraints and query workload can be shown to be intractable. We apply heuristic search techniques based on Greedy Hill Climbing to come up with nearly optimal solutions.

2 System Architecture

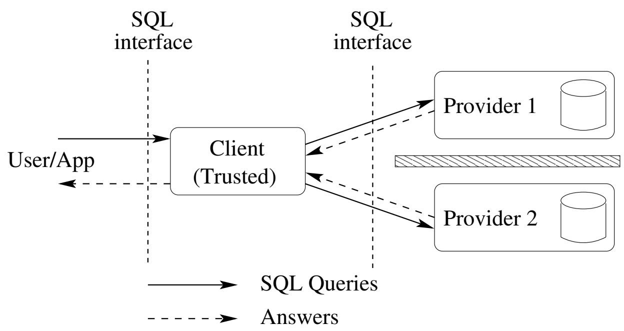

Figure 1. Distributed Architecture for a Secure Database Service

The general architecture of a distributed secure database service, as illustrated in Figure 1, is described more in [3]. It 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.

Some relevant terms are described here before going into further details.

- **Data Schema** This is the schema of the relation the client wishes to store on the server. As a running example, consider a company desiring to store relation R with the following schema.
  `R (Name, DoB, Gender, ZipCode, Position, Salary, Email, Telephone)`
- **Privacy Constraints:** These are described a collection of subsets of columns of a relation which should not be accessible together. The company may have the following privacy constraints defined:
  {`Telephone`}, {`Email`}, {`Name, Salary`}, {`Name, Position`},{`Name, DoB`}, {`DoB, Gender, ZipCode`},{`Position, Salary`}, {`Salary, DoB`}
- **Workload:** A workload W is a set of queries that will be executed on a regular basis on the client's data.
- **Tuple ID (TID):** Each tuple of the relation is assigned a unique tuple ID. The TID is used to merge data from multiple servers when executing a query on the data. The use of TID will become more explicit in the query plans described next.
- **Partitions:** There are two servers to store the client database. The schema and data is partitioned vertically and stored at the two servers.
  A partition of the schema can be described by three sets R_1 (attributes of R stored on Server 1), R_2 (attributes of R stored on Server 2) and E(set of encrypted attributes stored on both servers). It is important to note that $(R_1 \bigcup R_2 \bigcup E) = R$ and it is not necessarily the case that $R_1 \bigcap R_2 = \phi$. We denote a decomposition of R as $D(R)$.An example decomposition $D(R)$ of R is given here.
  `Partition 1 (`R_1`): (TID, Name, Email, Telephone, Gender, Salary)`
  `Partition 2 (`R_2`): (TID, Position, DoB, Email, Telephone, ZipCode)`
  `Encrypted Attributes (`E`): Email, Telephone`
- **Query Execution Plans in Distributed Environment:** When data is fragmented across multiple servers, there are two plan types used frequently to execute queries on data stored on these servers.
  Centralized Plans: On execution of a query, data from each server is transmitted to the client and all further processing is done at the client side. In some cases, multiple requests can go the each server but data from one server is never directly sent over to the other servers.
  Semi join Plans: As an alternative to centralized plans, it maybe more efficient to consider semi join plans. Here, $TIDs$ are passed from one server to the other to reduce the amount of traffic flow to the client.
- **Encryption Details:** Encryption of columns can either be deterministic or non-deterministic. A deterministic encryption is one which encrypts a column value k to the same value $E(k)$ every time. Thus, it allows equality conditions on encrypted columns to be executed on the server. Our implementation assumes encryption on columns to be deterministic.
- **Column Replication:** When columns of a relation are encrypted, then they can be placed in any of the two servers since they will satisfy all privacy constraints. It is beneficial to store the encrypted columns on both servers to make query processing more efficient. Non encrypted columns can also be duplicated as long as privacy constraints are also satisfied. Replication will result in lesser network traffic most of the time.
- **Cost Overhead:** We model the cost as the number of bytes transmitted on the network assuming that this supersedes the I/O cost on the servers and processing cost on the client. Cost overhead is the parameter used to determine the best possible partitioning of a relation. It measures the number of excess bytes transferred from the server to the client due to the partition.
  Cost Overhead($D(R)$) = $X - Y$,

where X = Bytes transmitted when executing workload W on a decomposition $D(R)$ of R at two servers,
Y = Bytes transmitted when executing workload W on relation R at one server with no fragmentation.

The problem can now formally be defined as follows.
We are given: (1) A data schema R; (2) A set of privacy constraints P over the columns of the schema R; (3) A workload W defined as a set of queries over R.; We have to come up with the best possible decomposition $D(R)$ of the columns of R into $R1$, $R2$ and E such that:
(1) All privacy constraints in P are satisfied. These can either be satisfied by encrypting one or more attributes in the constraint or have at least one column of the constraint at each of the servers. Encrypting columns has its disadvantages as discussed before so we give priority to splitting columns as a way to satisfy privacy constraints.
(2)The cost overhead of $D(R)$ for the workload W should be the minimum possible over all decompositions of R which satisfy P. Space is not considered as a constraint and columns of relations are replicated at both servers as long as they satisfy privacy constraints.

3 Intractability of Schema Decomposition

In this section, we provide hardness of approximation results for the schema decomposition problem. These results are not essential to understand the rest of the paper. They provide a formal reasoning why simplified versions of the schema decomposition problems are hard to approximate.
A standard framework to capture the costs of different decompositions, for a given workload W, is the notion of the *affinity matrix* [24] M, which we adopt and generalize as follows:

- The entry M_{ij} represents the performance "cost" of placing the unencrypted attributes i and j in different fragments.
- The entry M_{ii} represents the "cost" of encrypting attribute i across both fragments.

We assume that the cost of a decomposition may be expressed simply by a linear combination of entries in the affinity matrix. Let $R = \{A_1, A_2, \dots A_n\}$ represents the original set of n attributes, and consider a decomposition of $\mathcal{D}(R) = \langle R_1, R_2, E\rangle$, where R_1 is at Server 1, R_2 at Server 2 and E the set of encoded attributes. Then, we assume that the cost of this decomposition $C(\mathcal{D})$ using [3] is $\sum_{i\in(R_1-E),j\in(R_2-E)} M_{ij}$ + $\sum_{i\in E} M_{ii}$. (For simplicity, we do not consider replicating any unencoded attribute, other than the tupleID, at both servers.

In other words, we add up all matrix entries corresponding to pairs of attributes that are separated by fragmentation, as well as diagonal entries corresponding to encoded attributes, and consider this sum to be the cost of the decomposition.

Given this simple model of the cost of decompositions, we may now define an optimization problem to identify the best decomposition:

Given a set of privacy constraints $\mathcal{P} \subseteq 2^R$ and an affinity matrix M, find a decomposition $\mathcal{D}(R) = \langle R_1, R_2, E\rangle$ such that

(a) $\mathcal{D}$ obeys all privacy constraints in $\mathcal{P}$, and

(c) $\sum_{i,j:i\in(R_1-E),j\in(R_2-E)} M_{ij} + \sum_{i\in E} M_i$ is minimized.

We model the above problem with a graph theoretic abstraction. Each column of the relation is modeled as a vertex of the graph $G(V, E)$, whose edges weights are M_{ij} and vertex weights are M_{ii}. We are also given a collection of subsets $\mathcal{P} \subseteq 2^R$, say $S_1, \dots, S_t$ which model the privacy constraints. Given this graph $G(V, E)$ with both vertex and edge non-negative weights, our goal is to partition the vertex set V into three subsets - E (the encrypted attributes), R_1 (the attributes at Server 1) and R_2 (the attributes at Server 2). The cost of such a partition is the total vertex weight in E, plus the edge weight of the cut edges from R_1 to R_2. However, the constraint is that none of the subsets $S_1, \dots, S_t$ can be fully contained inside either R_1 or R_2. The closely related minimum graph homomorphism problem was studied in [5].

3.1 Minimum Cut when there are Few Sets S_i

There is an algorithm that solves the general problem, but this algorithm is efficient only in special cases, as follows. The proof for this theorem and all other theorems that follow can be found in [11], the extended version of the paper.

Theorem 1 *The general problem can be solved exactly in time polynomial in $\prod_i |Si| = n^{O(t)}$ by a minimum cut algorithm, so the general problem is polynomial if the S_i consist of a constant number of arbitrary sets, a logarithmic number of constant size sets, $O(\log n/\log\log n)$ sets of polylogarithmic size, and $(\log n)^\epsilon$ sets of size $e^{(\log n)^{1-\epsilon}}$ for a constant number of distinct $0 < \epsilon < 1$.*

3.2 Minimum Hitting Set when Solutions do not Use R_2

When edges have infinite weight, no edge may join R_1 and R_2 in a solution.

In the *hitting set problem* we are asked to select a set E of minimum weight that intersects all the sets in a collection of sets S_i.

Theorem 2 *For instances whose edges form a complete graph with edges of infinite weight the problem is equivalent to hitting set, and thus has $\Theta(\log n)$ easiness and hardness of approximation.*

3.3 The case $|S_i| = 2$ and Minimum Edge Deletion Bipartition

When vertices have infinite weight, no vertex may go to E.

In the *minimum edge deletion bipartition problem* we are given a graph and the aim is to select a set of edges of minimum weight to remove so that the resulting subgraph after deletion is bipartite. This problem is constant factor hard to approximate even when the optimum is proportional to the number of edges, as shown by Hastad [19], can be approximated within a factor of $O(\log n)$ as shown by Garg, Vazirani, and Yannakakis [12], and within an improved factor of $O(\sqrt{\log n})$ as shown by Agarwal et al. [2].

The next three results compare the problem having $|S_i| = 2$ to minimum edge deletion bipartition.

Theorem 3 *If all vertex weights are infinite (so that E may not be used), the sets S_i are disjoint and have $|S_i| = 2$, and all edge weights are 1, then the problem encodes the minimum edge deletion bipartition problem and is thus constant factor hard to approximate even when the optimum has value proportional to the number of edges.*

Theorem 4 *If all vertex weights are infinite (so that E may not be used), the sets S_i have $|S_i| = 2$, then the problem may be approximated in polynomial time by a minimum edge deletion bipartition instance giving an $O(\sqrt{\log n})$ approximation.*

Theorem 5 *If all vertex weights are 1, there are no edges, and the sets S_i have $|S_i| = 2$, then the problem encodes minimum edge deletion bipartition and is thus hard to approximate within some constant even for instances that have optimum proportional to the number of vertices.*

We now approximate the general problem with sets S_i having $|S_i| = 2$. The performance is similar to the minimum vertex deletion problem.

Theorem 6 *The general problem with sets S_i having $|S_i| = 2$ can be solved with an approximation factor of $O(\sqrt{n})$ by directed multicut.*

3.4 The case $|S_i| = 3$ and Intractability

The problem with $|S_i| = 3$ becomes much harder to approximate, compared to the $O(\sqrt{n})$ factor for $|S_i| = 2$.

Theorem 7 *If all vertex weights are 1, there are no edges, and the sets S_i have $|S_i| = 3$, then the problem encodes not-all-equal 3-satisfiability and it is thus hard to distinguish instances of zero cost from instances of cost proportional to the number of vertices.*

We examine the tractability when the sets S_i are disjoint.

Theorem 8 *If all vertex weights are infinite (so that E may not be used), the sets S_i are disjoint and have $|S_i| = 3$, and all edge weights are 1, then the problem encodes not-all-equal 3-satisfiability and it is thus hard to distinguish instances of zero cost from instances of cost proportional to the number of edges.*

Theorem 9 *The problem with vertices of infinite weight and edges of weight 1, sets S_i with $|S_i| = 3$ forming a partition with no edges within an S_i, the graph $H_{1,2,3,1',2',3'}$ with no edges joining $S = \{1, 2, 3\}$ and $S' = \{1', 2', 3'\}$ allowed, can be classified as follows:*

(1) If only additional H from K_0 are allowed, the problem is constant factor approximable;

(2) If only additional H from K_0 and K_1 are allowed, the problem is $O(\sqrt{\log n})$ approximable; furthermore as long as some graph from K_1 is allowed, the problem is no easier to approximate than minimum edge deletion bipartition, up to constant factors.

(3) If only additional H from K_0, K_1 and K_2 are allowed, the problem is $O(\log n)$ approximable;

(4) If some additional H from K_3 is allowed it is hard to distinguish instances with cost zero from instances with cost proportional to the number of edges.

We finally note that for dense instances with n vertices, m edges and sets S_i of constant size, we may apply the techniques of Alon et al. [7] to solve the problem within an additive $\epsilon \cdot m$ in time $2^{\tilde{O}(n^2/(\epsilon^2 m))} O(n^2)$ for $m = |E(G)|$.

4 Cost Estimation

Since algorithms with theoretical guarantees are hard to obtain for our data partition problem (Section 3), we develop instead 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 (Section 5) 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.

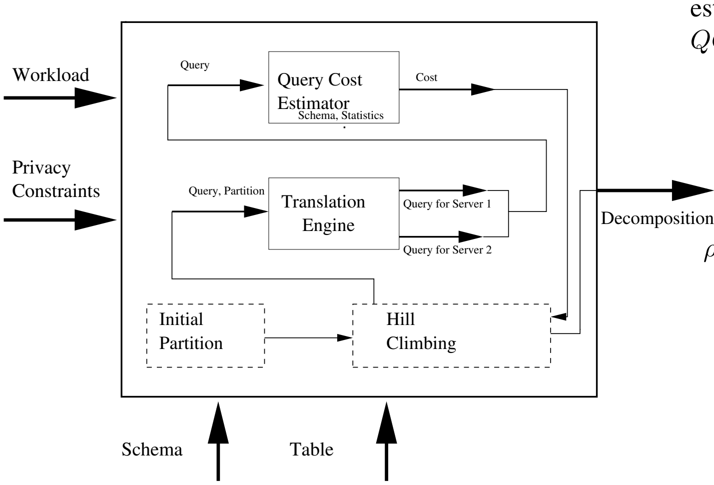

Figure 2. Components of the Partitioning Algorithm

4.1 Query cost estimator

We limit the type of queries as define by the grammar in Figure 3. The grammar specifies the valid predicates that we support as part of the WHERE clause of a SQL query.

$$
\begin{array}{rclcl}
 & & S & ::= & \langle \Sigma, a, c \rangle \\
 & & \Sigma & \in & \mathbb{P} \rightarrow \text{Predicate} \\
\text{Predicate} & \ni & F & ::= & (F_1 \wedge F_2) \mid (F_1 \vee F_2) \mid a = c \mid \\
 & & & & a \leq c \mid c \leq a \mid c_1 \leq a \leq c_2 \mid \\
 & & & & a_1 = a_2 \\
 & & a & \in & \mathbb{A} = \{\, \mathtt{T.x}, \mathtt{T.y}, ... \,\} \\
 & & c & \in & \mathbb{Z} = \{\, ..., -1, 0, 1, ... \,\}
\end{array}
$$

Figure 3. Syntax of the Boolean Predicate

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 in Figure 3. We use $\rho(F)$ to denote the selectivity of the formula F. $\rho(F)$ is computed recursively using the semantics given in Figure 4.

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.
$QC(q) = (\sum_{i \in SL(q)} S(R, i)) * T(R) * \rho(F)$

$$
\rho(F) = \begin{cases}
\frac{1}{V(a)} & \text{if } a = c \\
\frac{c - min(a) + 1}{max(a) - min(a) + 1} & \text{if } a \leq c \\
\frac{max(a) - c + 1}{max(a) - min(a) + 1} & \text{if } c \leq a \\
\frac{c_2 - c_1 + 1}{max(a) - min(a) + 1} & \text{if } c_1 \leq a \leq c_2 \\
\frac{1}{max(V(a_1), V(a_2))} & \text{if } a_1 = a_2 \\
1 - (1 - \rho(F_1))(1 - \rho(F_2)) & \text{if } F = F_1 \vee F_2 \\
\rho(F_1) \times \rho(F_2) & \text{if } F = F_1 \wedge F_2
\end{cases}
$$

Figure 4. Semantics of the Boolean Predicate

4.2 Translation 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 [24]. 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.

4.2.1 WHERE clause processing

The basic units of the WHERE clause are conditional clauses where operators could be $>$, $<$ or = as per the grammar defined above. Each of these basic units are combined using the AND or the OR operator. The entire clause itself can be effectively represented by a parse tree. Such a parse tree has operators as non-leaf nodes and operands as leaf nodes.

Bottom Up State Evaluation:
Bottom up evaluation of the parse tree starts at the leaf nodes. Each node transmits to its parent, its state information. The parent operator (always a binary operator) will combine the states of its left and right subtrees to generate a new state for itself.

State Definitions:
Each node in the tree is assigned a state value. Let W be the state value of the root of the parse tree. The semantics of the state value are as follows.

0: condition clause cannot be pushed to either servers; 1: condition clause can be pushed to Server 1; 2: condition clause can be pushed to Server 2; 3: condition clause can be pushed to both servers; 4: condition clause can be pushed to either servers.
As we proceed to determine the state values for all nodes, we need to consider nodes with state value 0 as a special case. All child attributes for a node with state value 0 are added to the select list of the query.
There are three cases to consider for a non-leaf node.
Case 1: The parent node is one of the operators $>$, $<$, =.
For such a parent node, the child nodes are attributes of relations or constant values. If the condition is an attribute-value clause, the state of the parent is the state value of the attribute. The state value of the attribute in turn is determined by the location of that attribute. If the condition is an attribute-attribute clause (a1 = a2 or a1 $<$ a2 etc), the state of the parent is:
0 if the state values of the attributes are (1,2) or if they are on the same server but one of the attributes is encrypted.; 1 if the state values of the attributes are (1,1) or (1,4) ; 2 if the state values of the attributes are (2,2) or (2,4) ; 4 if the state values of the attributes are (4,4)
Case 2: The parent node is the AND operator
Table 1 represents the state determination of an AND parent node given the state values of its two children. The row in the table is the state of the first child, the column is the state of the second child and the table entry is the state of the parent. For example, (4,3) implies a state 3 for the parent using the AND table.
Case 3: The parent node is the OR operator. Table 2 represents the state determination of an OR parent node given the state values of its two children.

Table 1. AND operator state table

	0	1	2	3	4
0	0	3	3	3	3
1	3	1	3	3	1
2	3	3	2	3	2
3	3	3	3	3	3
4	3	1	2	3	4

Table 2. OR operator state table

	0	1	2	3	4
0	0	0	0	0	0
1	0	1	0	0	1
2	0	0	2	0	2
3	0	0	0	3	3
4	0	1	2	3	4

4.2.2 SELECT clause processing

The select part of each query can be a set of attributes of the updated relations in the schema. The select clause processing generates a state S and two sets of attributes A1 and A2. S represents the following cases.
1: Requires access to Server 1 only; 2: Requires access to Server 2 only; 3: Requires access to both servers; 4: Requires access to any of the two servers.
A1 and A2 are the attributes in the select clause that are present on Server 1 and Server 2 respectively. If the attribute is present on both servers, it will be contained in A1 and A2.

4.2.3 Final Query Decomposition step

The final step is to generate the two queries Query 1 (to be sent to Server 1) and Query 2(to be sent to Server 2). We use the state value of the root node obtained from the WHERE clause processing W and the SELECT clause state value S.There are five cases depending on the state value of the root node .
W = 0:
There are no where clauses in Query 1 and Query 2 since none of the conditions can be pushed to the servers.
W = 1 or 2:
For state value 1, Query 2 does not contain any where clause.Similarly, for state value 2, Query 1 does not have any where clause.
W = 3 - Top Down processing of Clauses:
We perform a top-down processing of the tree. We start at the root operator and proceed downwards as long as we encounter state 3. We thus stop when we are sure whether to include the clause as part of Query 1 (state value 1), Query 2(state value 2) , Query 1/Query 2 (state value 4) or

not to include it at all(state value 0).
W = 4:
The where clauses can be pushed completely to any one of the two servers.

Let us work with an example to see how all this fits in to solve the problem as a whole. Consider a slightly more complicated client query.

```
SELECT Name, DoB, Salary FROM R
WHERE (Name ='Tom' AND
Position='Staff') AND (Zipcode =
'94305' OR Salary > 60000)
```

Let the predicates in the query be assigned P1 (Name = 'Tom'), P2(Position='Staff'), P3(Zipcode='94305') and P4(Salary>60000) The parse tree and the corresponding state values are shown in Figure 5.

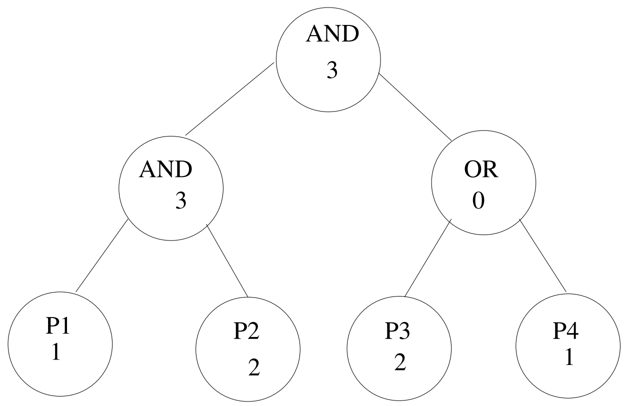

Figure 5. State Computation for the Predicate

```
  Query1:  SELECT TID, name, salary
FROM R1 WHERE Name='Tom'
Query2:  SELECT TID, dob, zipcode
FROM R2 WHERE Position='Staff'
```

The query plan is also shown here in Figure 6 detailing out the steps that need to be performed for executing this query. In the plan, T,N,S and D stand for TID, Name, Salary and DoB attributes of schema R respectively. At the client side, results of Query1 and Query2 are joined on attribute TID. The predicates P3 and P4 are then applied to the results followed by a projection on the select attributes.

5 Partitioning Algorithms

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.

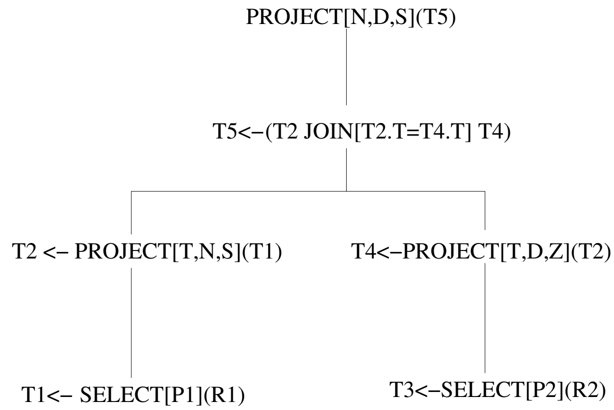

Figure 6. Distributed Query Plan

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 [27] 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 a 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 4^n 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 Experimental Results

6.1 Details of Experimental Setup:

We execute our code on a single relation R in all experiments. The number of attributes in R was varied from 1 to 30. The number of tuples in R was between 1000 and 10,000. Note that for our experiments we do not actually need the data, only the statistics that describe the data. Thus, the results we obtain for this setting are applicable to relations of a larger (or smaller) size.

As we vary the number of attributes, we generate privacy constraints over it randomly with the following properties. We generate as many privacy constraints as the number of attributes in the relation. Privacy constraints vary in size from one to the number of attributes in the relation. The attributes that are part of each constraint are selected at random from the available attributes without replacement.

Another important parameter is the workload. We generated 25 workloads containing a fixed number of queries(5 in our case) for different number of attributes of the relations. So, for a relation with fixed number of attributes, we generate about 125 queries (25*5) divided into twenty five workloads. For each query, the parts that were varied were the attribute set in the SELECT clauses and the conditions in the WHERE clause.We selected a subset of SQL which mapped to our grammar defined in Section 4. Each condition clause C was of the form $(xOPy)$ where OP was chosen at random to be $>$, $<$ or $=$. x's were chosen from the columns of R while y was chosen from the domain of x after choosing x. The $C's$ themselves were combined by choosing one of OR, AND. The other parameters that were randomly chosen were the number of condition clauses, the number of attributes in the SELECT clause and the actual attributes in the SELECT clause itself.

6.2 Synthetic Data Generation

We conduct experiments to demonstrate how well hill climbing compares with brute force. For each algorithm, we vary the number of attributes N from 1 to 6. We generate ten different workloads for each N and each workload was composed of 5 queries.While we can obtain results for hill climbing for larger N, the brute force approach starts to get intractable. Hill climbing starts to move away from the optimal solution with increasing N. For 3 attributes, hill climbing is around 20 percent away from brute force but for 6 attributes, this percentage goes up to around 140.

Next, we study the behavior of hill climbing in terms of the number of iterations it takes to converge. We vary the number of attributes N for the relation from 1 to 30 for these experiments.The number of workloads for each N is 10 and similar to the previous experiment, we have 5 queries per workload.

Figure 7 shows the number of workloads for which the hill climbing converged . We note that the number of workloads requiring more than 10 iterations is less than 2 percent of the workloads.

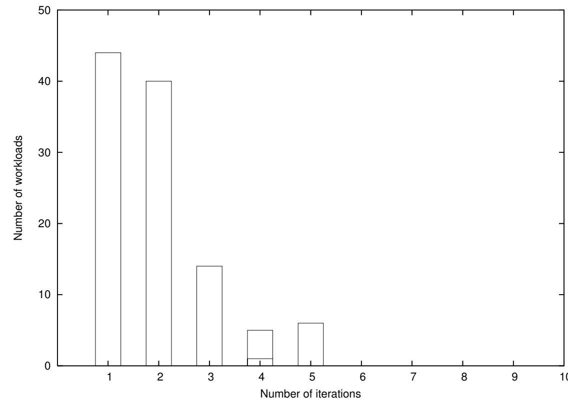

Figure 7. Hill Climbing Iterations

We now show the improvements achieved by the hill climbing over the initial partition. The improvement percentage is defined as:

`Perc. Improvement =` $(C_{fp} - C_{ip})/C_{ip}$ `* 100`

where C_{fp} = Cost estimate for the final partition of R returned by Hill Climbing for a given workload W and set of privacy constraints P; and C_{ip} = Cost estimate of the initial partition given as input to the hill climbing algorithm for the same workload W and set of privacy constraints P. Figure 8 illustrates that the number of workloads with greater than 20 percent difference from the initial solution keeps decreasing with increasing percentages. Despite this fact, more than 50 percent of the workloads have a percentage improvement of over 25 percent.

6.3 Personal Data Example

We run experiments for the real world example discussed earlier in the paper in Section 2. The schema R is the same with 8 attributes and 8 privacy constraints. The workloads are generated at random as discussed in the previous subsection. Figure 9 depicts that for about 50 percent of the workloads, the percentage difference of the final result of hill climbing from the initial partition is about 5 percent. Thus, given privacy constraints of the form listed for this example, it is easier to guess a reasonably valid and optimal decomposition for the schema. We also find that hill climbing terminates sooner (fewer number of iterations as compared to the synthetic data case) with close to 50 percent of the workloads terminating in a single iteration.

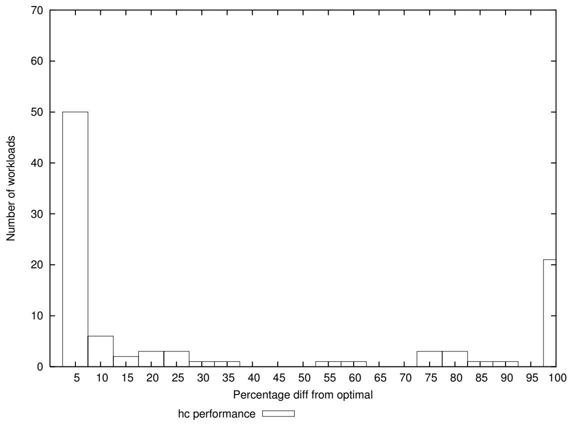

Figure 8. Performance Gain using Hill Climbing

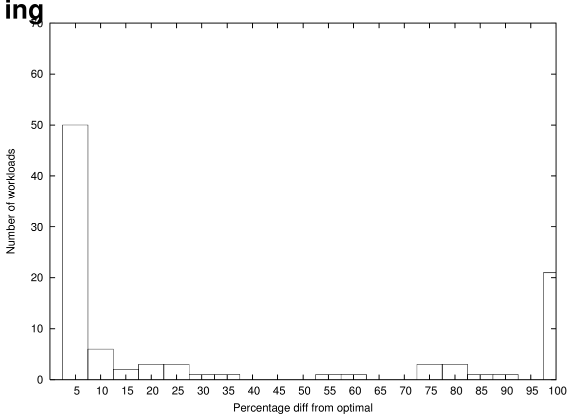

Figure 9. Personal Data Example - Performance Gain using Hill Climbing

6.4 Varying Distributions of Privacy Constraints Generation Task

We had previously generated a synthetic dataset for the privacy constraints and the workload where all parameters were generated uniformly at random. For this experiment, we use a relation with 8 columns and 1000 tuples. There are 8 privacy constraints generated for the relation. There is a percentage weight parameter governing the generation of constraints which applies to the first three columns of the relation. For example, if percentage weight is set to 20, then 60 percent (20*3) of the time, one of the first three columns will be selected as a participant in the generated privacy constraint. So, setting this weight to around 12 results in a uniform distribution and as it goes above 33, we get a heavily biased set of privacy constraints on these three columns. We generate 30 workloads for each weight value that we desire to test. Figure 10 shows the average number of iterations as we vary the weight from 10 to 34 percent. It can be seen that as the bias on the three attributes increases, we get to the final result in fewer number of iterations. Fewer iterations are required because most of the privacy constraints are concerned with these three attributes and are independent of the others so we have fewer options to choose for these three attributes.

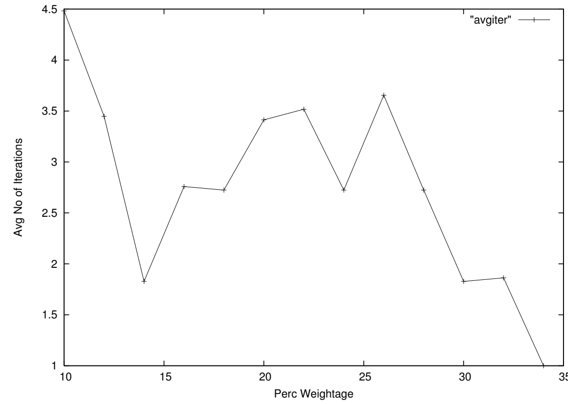

Figure 10. Average Iterations with varying privacy constraints distribution

7 Related Work

There is a wide consensus that privacy is a corporate responsibility [21]. In order to help and ensure corporations fulfill this responsibility, governments all over the world have passed multiple privacy acts and laws, for example, Gramm-Leach-Bliley (GLB)Act [14], Sarbanes-Oxley (SOX) Act [26], Health Insurance Portability and Accountability Act (HIPAA) [20], SB1386 [1] are some such well known U.S. privacy acts. In many use cases complying with these laws require an organization to encrypt the data in case it is hosted by an external service provider.

As discussed in the introduction, the outsourcing of data management has motivated the model where a DBMS provides reliable storage and efficient query execution, while not knowing the contents of the database [17]. Schemes proposed so far for this SaaS model [16] (Software as a Service) encrypt data on the client side and then store the encrypted database on the server side [16, 18, 6]. However, in order to achieve efficient query processing, all the above schemes only provide very weak notions of data privacy. In fact a server that is secure under formal cryptographic notions can be proved to be hopelessly inefficient for data processing [22].

After our original work [3] on using a combination of distribution and encryption for secure databases, our colleagues at Stanford University have developed partially [9] and complete homomorphic encryption algorithms [13].

This allows arithmetic operations on ciphertext without decrypting to plaintext. Homomorphic encryption along with deterministic hashing primitives drastically improves SQL processing on encrypted data in the SaaS model.

[3, 23, 10] define privacy constraints and describe the architecture for secure database services. In this paper, we build on this architecture and develop algorithms to split the schema among the servers and perform distributed query processing for a subset of SQL (both are absent in [23]). K-anonymity [4] maintains aggregates while changing microdata for data publishing. This scheme maintains microdata intact for OLTP workloads also.

References

[1] *California Senate Bill SB 1386*, Sept. 2002.

[2] A. Agarwal, M. Charikar, K. Makarychev, and Y. Makarychev. $o(\sqrt{\log n})$ approximation algorithms for min uncut, min 2CNF deletion, and directed cut problems. In *Proc. 37th Ann. ACM STOC*, pages 573–581, 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] 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.

[5] G. Aggarwal, T. Feder, R. Motwani, and A. Zhu. Channel assignment in wireless networks and classification of minimum graph homomorphism. In *ECCC TR06-040*, 2006.

[6] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order-preserving encryption for numeric data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2004.

[7] N. Alon, W. F. de la Vega, R. Kannan, and M. Karpinski. Random sampling and approximation of max-CSP problems. In *J. Comput. Syst. Sci.*, pages 212–243, 2003.

[8] Amazon. Amazon elastic compute cloud. Available from URL: http://www.amazon.com/b/ref-=sc_fe_l_2/?node=201590011&no=3435361.

[9] D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-dnf formulas on ciphertexts. In *Proc. TCC*, pages 325–341, 2005.

[10] V. Ciriani, S. D. C. di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Fragmentation and encryption to enforce privacy in data storage. In *ESORICS*, pages 171–186, 2007.

[11] V. Ganapathy, D. Thomas, T. Feder, H. Garcia-Molina, and R. Motwani. Distributing data for secure database services. Technical report, Stanford University, 2007. Available from URL:http://dbpubs.stanford.edu/pub/2007-23.

[12] N. Garg, V. Vazirani, and M. Yannakakis. Approximate max-flow min-(multi)cut theorems and their applications. In *SIAM J. Comp.*, pages 235–251, 1996.

[13] C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proc. STOC*, pages 169–178, 2009.

[14] GLB. Gramm-Leach-Bliley Act. Available from URL: http://www.ftc.gov/privacy/privacyinitiatives/glbact.html.

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

[16] 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.

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

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

[19] J. Hastad. Some optimal inapproximability results. In *Proc. 29th Annual ACM Symp. on Theory of Computing*, pages 1–10, 1997.

[20] HIPAA. Health Information Portability and Accountability Act. Available from URL: http://www.hhs.gov/ocr/hipaa/.

[21] IBM. Privacy is good for business. Available from URL: http://www-306.ibm.com/innovation/us/customerloyalty/harriet_pearson_interview.shtml.

[22] M. Kantarcioglu and C. Clifton. Security issues in querying encrypted data. In *19th Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, 2005.

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

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

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

[26] SOX. Sarbanes-Oxley Act. Available from URL: http://www.sec.gov/about/laws/soa2002.pdf.

[27] V. Vazirani. *Approximation Algorithms*. Springer, 2004.