Partitioning Columns For Secure Distributed Databases

Tomás Feder
Stanford University
tomas@theory.stanford.edu

Vignesh Ganapathy
Stanford University
vignesh@cs.stanford.edu

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

Rajeev Motwani
Stanford University
rajeev@cs .stanford.edu

Dilys Thomas
Stanford University
dilys@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 was proposed as a solution to this problem. In this paper we explore partitioning algorithms for these distributed architectures. We show that a general version of the problem is NP-hard.We therefore propose heuristic techniques based on greedy hill climbing to obtain partitions which satisfy the privacy constraints. Experimental results show that in practice our heuristics provide nearly optimal solutions.

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 [6],[9],[20]. As a result of such service providers, parties wanting to store and manage their data prefer outsourcing data to these service providers. The parties who outsource their data will be referred to as 'clients' hereafter. The server machines of service providers storing data will be referred to as 'servers' or 'sites'. Also, the database outsourced by the client will be considered to be a single relation for the purpose of this paper. Our results naturally extend to the case of a schema with multiple relations.

There is a growing concern regarding data privacy among clients. Often, client data has sensitive information which they do not want to compromise. 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 [2], [18]. 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.

VLDB '07, September 23-28, 2007, Vienna, Austria.

Some of the existing solutions for data privacy involve encryption of the entire database when storing them on the server. A client query request requires the entire database to be transferred to the client and decrypted to get the result. The previous solution has the obvious disadvantage of heavy network traffic as well as decryption cost for each query. Improvements include encrypting individual sensitive columns instead of the entire database so most queries do not require the entire relation to be transmitted to the client.

[2] 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 sites 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.

1. None of the privacy constraints should be violated.
2. For a given workload, minimum number of bytes should be transferred from the servers to 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 when modeled as a graph. We apply heuristic search techniques based on Greedy Hill Climbing to come up with nearly optimal solutions.

Layout: The next section explains relevant terminology and system architecture for such a database service. The intractability of the problem is described in detail in Section 3. The hill climbing technique is then discussed in detail followed by experimental setup and results. We then conclude with related and future work in this direction.

2. SYSTEM ARCHITECTURE

The general architecture of a distributed secure database service, as illustrated in Figure 1 is described more in [2]. 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.

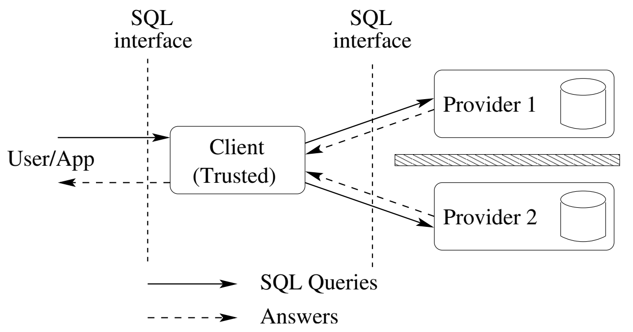

Figure 1: Distributed Architecture for a Secure Database Service

Some relevant terms are described here before going into further details. 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 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 is a set of queries that will be executed on a regular basis on the client's data.A possible workload on R could be:
  ```
  select Name from R where Position = 'Staff';
  select * from R where Salary > 90,000;
  select Name, Email, Telephone from R where
  Gender = 'F' and ZipCode = '94305';
  ```

- **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. This will become more explicit in the query plans described next.

- **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.Consider the following fragmentation of the relation R.
  ```
  Partition 1 (R1): (TID, Name, Email,
  Telephone, Gender, Salary)
  Partition 2 (R2): (TID, Position, DoB, ZipCode)
  Query: select * from R where Salary>90,000;
  ```
  Assume salary is not encrypted. The query is split into the following queries:
  ```
  Query 1: select TID, Name, Email, Telephone,
  Gender, Salar from R1 where Salary > 90,000
  Query 2: select TID,Position,DoB,ZipCode from R2.
  ```
  The results with matching TIDs from both queries are returned as the final result.
  Semijoin Plans: As an alternative to centralized plans, it is more efficient to consider semijoin plans. Here, TIDs are passed from one server to the other to reduce the amount of traffic flow to the client. For the same example query, the first query is:
  ```
  Query 1: select TID, Name, Email, Telephone,
  Gender, Salary from R1 where Salary > 90,000
  ```
  The result of Query 1 is returned to the client and the TIDs are passed to server 2 to get the matching tuples from R2.
  ```
  Query 2: select TID, Position, DoB,
  ZipCode from R2 where TID in <TIDs returned from
  Query 1>.
  ```

- **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. Replication will result in lesser network traffic most of the time. Our implementation assumes all encrypted columns to be available on both servers.

- **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.

- **Cost Overhead:** As in [2] 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 = (Bytes transmitted when executing workload W on a fragmentation F of R at two sites) - (Bytes transmitted when executing workload W on relation R at one site with no fragmentation).`

3. INTRACTABILITY

A standard framework to capture the costs of different decompositions, for a given workload W, is the notion of the *affinity matrix* [19] M, which we adopt and generalize as follows:

1. The entry M_{ij} represents the "cost" of placing the unencoded attributes i and j in different fragments.

2. The entry M_{ii} represents the "cost" of encoding 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 $R = \{A_1, A_2, \ldots A_n\}$ represents the original set of n attributes, and consider a decomposition of $\mathcal{D}(R) = \langle R_1, R_2, E\rangle$. Then, we assume that the cost of this decomposition $C(\mathcal{D})$ 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 sites.)

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 of V, say $S_1, \ldots, S_t$ which model the privacy constraints. Given this graph $G(V, E)$ with both vertex and edge nonnegative weights, our goal is to partition the vertex set V into three subsets - say A (the encrypted attributes), B (the attributes at site 1) and C (the attributes at site 2). The cost of such a partition is the total vertex weight in A, plus the edge weight of the cut edges from B to C. However, the constraint is that none of the subsets $S_1, \ldots, S_t$ can be fully contained inside either B or C.

The closely related minimum graph homomorphism problem was studied in [3].

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.

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 conist 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 $0 < \epsilon < 1$.*

PROOF. One may try in all possible ways to select for each S_i either one element to go to A or two elements to go to B, C respectively. Merge together the identified elements of each of A, B, C next, remove the identified elements in A, and now we are looking for a min cut between the identified B, C elements, where the cut is given by vertices that go to A and edges that join B, C. In fact we may turn the edges into vertices by putting a middle vertex of the appropriate weight on each edge, so we are looking for a min-vertex cut, which is polynomial. Thus the complexity is polynomial in $\prod_i |S_i| = n^O(t)$ because of the initial number of possible choices. □

3.2 Minimum Hitting Set when Solutions do not Use C

When edges have infinite weight, no edge may join B and C in a solution.

In the *hitting set problem* we are asked to select a set A 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 (so that B, C may not both be used), the problem is equivalent to hitting set, and thus has $\Theta(\log n)$ easiness and hardness of approximation.*

PROOF. We may not cross the cut B, C as this would give infinite cost. We may thus assume that only A, B will be used. Each set S_i must then have at least one element in A, so a solution is valid only if A hits all the sets S_i, and the cost is the sum of the vertex weights in the hitting set A. □

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

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

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 [14], can be approximated within a factor of $O(\log n)$ as shown by Garg, Vazirani, and Yannakakis [7], and within an improved factor of $O(\sqrt{\log n})$ as shown by Agarwal et al. [1].

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 A 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.*

PROOF. Encode each vertex v of G as $S_v = \{a_v, b_v\}$, and encode each edge vw of G as the two edges joining S_v, S_w given by a_vb_w, b_va_w. The side of the bipartition for v depends on whether a_v or b_v goes to B, and an edge vw is removed if v, w go to the same side, in which case we pay for both a_vb_w, b_va_w across the cut B, C. Thus the problem is equivalent to the minimum edge deletion bipartition. □

THEOREM 4. *If all vertex weights are infinite (so that A 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.*

PROOF. If two sets S_i share an element, say $S_i = \{a, b\}$ and $S_j = \{a, c\}$, then we may merge b and c. We may thus assume the S_i are disjoint. Now represent $S_v = \{a_v, b_v\}$ by a vertex v, and if S_v, S_w are joined by edges we must pay at least one for these edges, unless these edges are (1) contained in a_vb_w, b_va_w or (2) contained in a_va_w, b_vb_w. In case (1) we join vw by an edge, and in case (2) we introduce a new vertex u and form a path vuw of length 2. This encodes the problem as a minimum edge deletion bipartition problem up to a costant factor, so an $O(\sqrt{\log n})$ approximation exists. The problem with edge weights is similarly solved by subtracting weights joining S_i and S_j until we fall in cases (1) or (2) above.

□

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.*

PROOF. If we consider the sets S_i as edges, this is the problem of removing the least number of vertices to make the graph bipartite. We know hardness for removing edges to make the graph bipartite. To translate to vertices, separate each vertex of degree d into d vertices, one for each adjacency, and connect these d vertices with a constant degree expander graph, where each edge of the expander graph is replaced with constant number of parallel paths of length two. Thus if less than half the vertices of the same expander graph get removed, this corresponds to removing a proportional number of edges. □

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.*

PROOF. Represent each vertex v of weight x by four vertices a_v, b_v, c_v, d_v joined by arcs $a_v b_v, c_v d_v$ of capacity x. Represent each $S_i = \{u, v\}$ by arcs $b_u c_v, d_u a_v, b_v c_u, d_v a_u$ of infinite capacity. Represent each edge uv of weight y by arcs $b_u a_v, d_u c_v, b_v a_u, d_v c_u$ of capacity y. Finally look for a multicut that separates the sources a_v from the corresponding sinks d_v. Removing an arc for a vertex x corresponds to assigning x to A, and after removing arcs corresponding to uv, the vertices form by reachability two components B for arcs $a_v b_v$ and C for arcs $c_v d_v$. The multicommodity flow result of Gupta [10] for directed multicut gives the $O(\sqrt{n})$ bound. □

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.*

PROOF. If there are are no edges and the sets S_i have size $|S_i| = 3$, then the problem encoded is not-all-equal 3-satisfiability by corrresponding sets B and C to values 0 and 1 respectively. Even satisfiable instances of not-all-equal 3-satisfiability have a constant factor hardness on the number of variables participating in unsatisfied clauses by a solution. We conclude that a constant fraction of such variables must be assigned to A even if a zero cost solution exists. The hardness of approximation of non-all-equal satisfiability for number of variables instead of clauses is obtained by making multiple copies of the same variable for multiple clauses, and joining these with a constant degree expander graph. Each edge xy of the expander graph represents a path of length two xzy, where xz and zy represent $x \neq z$ and $z \neq y$ over $\{0, 1\}$ respectively. We represent $x \neq y$ with clauses $\{x, t, y\}, \{x, u, y\}, \{x, v, y\}, \{t, u, v\}$. The result thus folows from the result for not-all-equal satisfiability of Hastad [14]. □

We examine the tractability when the sets S_i are disjoint.

THEOREM 8. *If all vertex weights are infinite (so that* A *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.*

PROOF. A clause of not-all-equal satisfiability may be viewed as a set $S_i = \{x_i, y_i, z_i\}$. We assume these S_i are disjoint and join copies of variables in different clauses by a clique. If the not-all-equal 3-satisfiability problem has a solution, a solution of zero cost exists for our problem. The number of clauses satisfied in a not-all-equal satisfiability problem is constant factor hard to approximate even on instances that are satisfiable. Therefore in a solution to our problem a constant fraction of the sides chosen for the elements of the S_i would have to be changed between B and C to obtain a consistent solution to not-all-equal 3-satisfiability that fails a constant fraction of the clauses. If we replace each clique by a constant degree expander graph, then each of the elements of S_i that would be changed between B and C pays a constant, as at most half of the elements of the expander graph for a clique are changed. Thus we pay cost proportional to the number of edges when the optimal cost is zero by hardness of approximation of not-all-equal 3-satisfiability shown by Hastad [14]. □

To obtain approximable cases with $|S_i| = 3$, we continue to consider the case where vertex weights are infinite and edge weights are 1, still assume the S_i form a disjoint partition of the vertex set without edges joining vertices in the same S_i, but limit the possible connections joining two sets S_i, S_j. We shall always allow two such sets not to be joined, and shall always tolerate a constant factor approximation. Thus if $S = \{1, 2, 3\}$ and $S' = \{1', 2', 3'\}$, all that matters is the connected components in the bipartite graph H joining S and S'. We may list such H by indexing them according to connected components in H, obtaining the following possibilities grouped into four sets: $K_0 = \{H_{1231',2',3'}, H_{1231'2',3'}, H_{1231'2'3'}, H_{121'2',3,3'}, H_{121'2',33'}\}$, $K_1 = \{H_{121',33',2'}, H_{121',32'3'}\}$, $K_2 = \{H_{11',22',33'}\}$,
$K_3 = \{H_{11',2,2',3,3'}, H_{11',22',3,3'}, H_{11'2',2,3,3'}\}$.
The problem gets gradually harder as we add possible patterns joining sets S_i with $|S_i| = 3$ in the order K_0, K_1, K_2, K_3.

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.

PROOF. We prove (4). The case of $H_{11',2,2',3,3'}$ was proven in the preceeding theorem. The case of $H_{11',22',3,3'}$ simulates $H_{11',2,2',3,3'}$ by considering $H_{11'',22'',3,3''}, H_{1''\hat{1},3''\hat{3},2'',\hat{2}}, H_{\hat{1}1',\hat{2}2',\hat{3},3'}$. The case of $H_{11'2',2,3,3'}$ simulates $H_{11',2,2',3,3'}$ by considering $H_{11''2'',2,3,3''}, H_{1''2''1',2',3',3''}$ This proves (4).

We next prove (1). Any occurrences of $H_{1231',2',3'}, H_{1231'2',3'}, H_{1231'2'3'}$ must pay in a solution, so we may remove these and pay cost proportional to the number of these. The graphs $H_{121'2',3,3'}, H_{121'2',33'}$ are equivalent up to constant factors, so we consider just $H_{121'2',33'}$. If 12 are combined in $H_{121'2',33'}$ and 13 are combined in $H_{131''3'',22''}$, then at least one of these two graphs must pay, so for each set $S = \{1, 2, 3\}$ we may consider the number a_{12} of graphs combining 12, the number a_{13} of graphs combining 13, and the number a_{23} of graphs combining 23, and remove the least two of a_{12}, a_{13}, a_{23} number of graphs (say remove the graphs for combinations 13 and 23 and keep the graphs for combinations 12). This incurs another constant factor of the optimum. Finally every S has only one combination 12, so we may assign 12 to B and 3 to C at zero cost. This proves (1).

We next prove (2). Define $\hat{H}_{11',22',3,3'}$ by $H_{11'',22''3'',3}, H_{1'1'',2'2''3'',3'}$. Thus $\hat{H}_{11',22',3,3'}$ is $H_{11',22',3,3'}$ plus the condition that $11'$ and $22'$ go to B and C respectively or to C and B respectively. We may remove occurrences of the first three graphs H in K_0 by paying cost proportional to the number of such H as before. The last two H in K_0 and the two H in K_1 can be simulated by $\hat{H} = \hat{H}_{11',22',3,3'}$, as they are superpositions of several copies of $\hat{H}$ under various permutations of the elements of S and S', and superpositions may be avoided by concatenating two copies of $\hat{H}$ to obtain $\hat{H}$ again. We may thus suppose that $\hat{H}$ is the only graph that occurs. Suppose the role of 3 in $\hat{H}$ for $S = \{1, 2, 3\}$ is played in different groups by $1, 2, 3$, so that we have $\hat{H}_{11',22',3,3'}$, $\hat{H}_{11'',33'',2,2''}$, $\hat{H}_{2\hat{2},3\hat{3},1,\hat{1}}$. Then one of these three must pay, so we may remove for S the one group that occurs in the least number of such graphs (say keep the first to with the role of 3 played by 2 or 3 and remove the ones where the role of 3 is played by 1). The cost payed is proportional to the number of such graphs removed, incurring a constant factor approximation. We may finally assume that only $\hat{H}_{12',21',3,3'}$, $\hat{H}_{13',31',2,2'}$ occur. If $\hat{H}_{11',22',3,3'}$ occurs, it can be simulated as $\hat{H}_{12'',21'',3,3'}$, $\hat{H}_{1'2''',2'1'',3,3'}$. Thus we may say that 1 crosses in such occurrences with either 2 or 3, and so the instance can be solved at zero cost if and only if the graph whose vertices are the S_i and the edges are the $\hat{H}_{12',21',3,3'}$ joining them is bipartite. The problem thus reduces to the minimum edge deletion bipartition problem and is solvable in time $O(\sqrt{\log n})$. It can be shown that as long as some graph in K_1 is allowed, the problem is no easier than minimum edge deletion bipartition, up to constant factors. This proves (2).

We finally prove (3). Suppose first only $H_{11',22',33'}$ occurs. This graph permutes 123 into $1'2'3'$ in some way. We may compose such permutations and come back to $S = \{1, 2, 3\}$. If 123 comes back as 231, then the instance has no solution of zero cost. If this never happens, say 123 only comes back as 213, then we may map 1, 2 to B and 3 to C, obtaining a solution of zero cost. We may thus represent each such S by six vertices corresponding to the six permutations $123, 132, 213, 231, 312, 321$, and match the six permutations for $S = \{1, 2, 3\}$ to the six permutations for $S' = \{1', 2', 3'\}$. On this graph with six vertices for each S, we may look for a multicut separating each pair $123, 231$ for each S. This can be done by the algorithm of Garg, Vazirani, and Yannakakis [7] with an $O(\log n)$ approximation. This proves the case of $H_{11',22',33'}$ alone.

We complete the proof of (3). The first three sets H of K_0 pay as from before and are removed at a constant factor approximation. The last two sets H of K_0 and the two sets H of K_1 can be simulated by $\hat{H}_{11',22',3,3'}$ defined as from before. The part of the problem involving only $H_{11',22',33'}$ can be solved with an $O(\log n)$ approximation, by removing the corresponding multicut. Now if a set S is connected directly to d other S_i, then create d copies of S, one for each S_i, and join the copies of S with a constant degree expander graph involving edges between this copies having $H_{11',22',33'}$. Finally for each occurrence of $\hat{H}_{11',22',3,3'}$ at $S = \{1, 2, 3\}$ (there is now at most one such occurrence at S) ask for a multicut separating 1, 2 for such S, as 1, 2 must go to B, C respectively or C, B respectively. This is again done with an $O(\log n)$ approximation by the algorithm of Garg, Vazirani, and Yannakakis [7]. This completes the proof of (3). □

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. [5] 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. HILL CLIMBING

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 result increases and then stopping when the result decreases. Hill-climbing converges to a local maxima.

Given a schema R, there are different ways to partition the attributes across the servers which satisfy the privacy constraints P. Given also a query workload W, there are some partitions which are more efficient than others in terms of number of bytes transmitted across the network. The problem is to find the best possible partition which minimizes the number of bytes transmitted across the network.

The problem is non trivial because of the exponential number of partitions that need to be checked for efficiency.

4.1 Brute Force Approach

Each column of the relation has three options for storage. (1) Store un-encrypted at site 1. (2)Store un-encrypted at site 2. (3) Store encrypted at both servers. The brute force essentially picks the best partition which satisfies all constraints and results in minimum network traffic. Thus, for a relation with n columns there are 3^n possible fragmentations possible and very few of them will satisfy all the privacy constraints. We therefore explore more efficient algorithms for the problem.

4.2 Hill Climbing Approach

An initial fragmentation of the database is considered which satisfies all the privacy constraints.

Initial Guess: The initial guess used as a starting point for the hill climbing algorithm decides how good the final result will be. The initial guess/state is a valid partition of relation R into R1 and R2 which satisfies all the privacy constraints. A good initial state is obtained using the weighted set cover.

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 such as:

1. Decrypting an encrypted column and placing it at site 1.
2. Decrypting an encrypted column and placing it at site 2.
3. Encrypting an un-encrypted 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.

This continues till the fragmentation does not change after a particular iteration. The initial solution decides the final result and hill climbing suffers from the issue of local minima so there are cases when the results from hill climbing do not match the brute force approach.

5. COST ESTIMATION

The cost overheads can be estimated based on the data collected by standard query optimizers in a database management system.

5.1 Centralized Plans with Deterministic Encryption

We first consider centralized plans with deterministic encryption. Consider the following query on R where $a1 \in R1$.
`Select A from R where a1 OP v1`
Consider the following quantities which are generally available with most query optimizers.
T(R): Number of tuples in R (including size of TupleID)
S(R): Number of bytes per tuple in R
B(R): Number of blocks to hold all tuples in R
V(R,A): Number of distinct values of A in R
p(cond(S)): Selectivity of condition on S
p(cond(S)) = $(T(R)/V(R,S))$,for equality conditions
p(cond(S)) = $v - min(S)/((max(S) - min(S))$, for $\leq$
p(cond(S)) = $max(S) - v/(max(S) - min(S))$, for $\geq$
S(R,S): Size in bytes of S in R

The computation for cost overhead for the above query is:
For the cost overhead estimates below, it is assumed that we have the equality operator if the attribute is encrypted. Thus, conditions on single attributes can be executed on the server.

If there are inequality predicates in the query on an encrypted attribute, these conditions must be executed on the client.

Single Relation R:
Total number of tuples in R, t = T(R)
Tuples that satisfy the AND predicate, $n = t * p(cond(a1))$
Total number of bytes transmitted $= n * (\sum_{i \in A} S(R,i))$

Partitioned Relation R: The query is split into the following queries for the two relations R1 and R2.

```
Select A1 from R1 where a1 OP v1
Select A2 from R2
```

Where $A1 \in (R1 \cap A)$ and $A2 \in (R2 \cap A)$
(If attributes are replicated at both sites, the projections and selections are pushed to the site which reduces the network traffic).
Total number of tuples in R1 and R2 = T(R)
Tuples that satisfy the AND predicate in R1, $n = T(R) * p(cond(a1))$
Total number of bytes transmitted for R1, $b1 = n * (\sum_{i \in (A \cap R1)} S(R,i)$
Total number of bytes transmitted for R2, $b2 = T(R) * (\sum_{i \in (A \cap R2)} S(R,i)$
Cost overhead = (Excess bytes transmitted from tuples in R2 that do not satisfy the AND predicate)
$= T(R) * (1 - p(cond(a1))) * (\sum_{i \in (A \cap R2)} S(R,i))$
We can come up with similar estimates for other kind of queries involving AND and OR predicates on the schema.

5.2 Semi join Plans with Deterministic Encryption

Consider the same query (as discussed for the centralized case) on R where $a1 \in R1$.
`select A from R where a1 OP v1`
The cost overhead computation for this query is described here.
Single Relation R:
Total number of tuples in R = T(R)
Tuples that satisfy the AND predicate, n = $T(R) * p(cond(a1))$
Total number of bytes transmitted, b = $n * \sum_{i \in A} S(R,i)$
Partitioned Relation R: The query is split into the following queries for the two relations R1 and R2. `select TID,A1 from R1 where a1 OP v1`

tt select A2 from R2 where TID = $< TIDreturnedfrompreviousquery >$

Where $A1 \in (A \cap R1)$ and $A2 \in (A \cap R2)$
(If attributes are replicated at both sites, the projections and selections are pushed to the site which reduces the network traffic) From the queries on R1 and R2, it is clear that the only extra overhead in executing the query is the TIDs which pass around from one server to the other.

Total number of tuples in R1 and R2 = T(R)
Tuples that satisfy the AND predicate in R1, n =

$T(R) * p(cond(a1))$
Total number of bytes of TI transmitted = $n * SizeOf(TID$
Cost overhead = $T(R) * p(cond(a1)) * SizeOf(TID)$

Semijoin plans result in much lesser network bandwidth as compared to centralized plans. We can come up with similar estimates for more complicated queries.

6. EXPERIMENTAL RESULTS

6.1 Details of experimental data:

The schema R: Name, Date of Birth (DoB), Gender, ZipCode, Position, Salary, Email, Telephone

Data in R: 10 tuples with different values for name, DoB, email and telephone for each tuple, , 70

Workload W: Following queries with combinations used for additional workloads. Overall, around 25-30 workloads were executed on this sample dataset.

Privacy Constraints P: Telephone, Email, Name, Salary, Name, Position, Name, DoB, DoB, Gender, ZipCode, Position, Salary, Salary, DoB

6.2 Hill Climbing Algorithm Behavior

Most of the queries took 2-3 iterations to converge to the final partition result. A very small number of them took three iterations to converge as shown in the pie chart below.

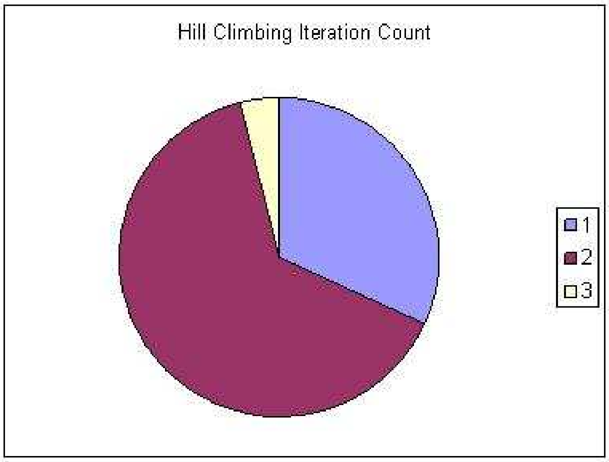

Figure 2: Hill Climbing Iterations

Hill Climbing Vs Brute Force

The performance of hill climbing as compared to Brute force is very good overall. Around 80-85Around 1

The points where these two techniques do not match shows a very large difference in the cost overheads. Thus, the hill climbing returns a partition with a very high cost overhead when it does not match the result of brute force.

Hill Climbing Vs number Encrypted Attributes

It is interesting to see how the difference in cost overhead relates to the number of encrypted attributes. Since only 1The difference in the number of encrypted attributes has also been plotted here.

The non-zero values for cost overhead difference are clearly visible. For most of the cases, the no. of encrypted attributes > 0 i.e. hill climbing encrypts more attributes than required. It is important to note that in three of the four cases, the hill climbing encrypts lesser attributes (difference

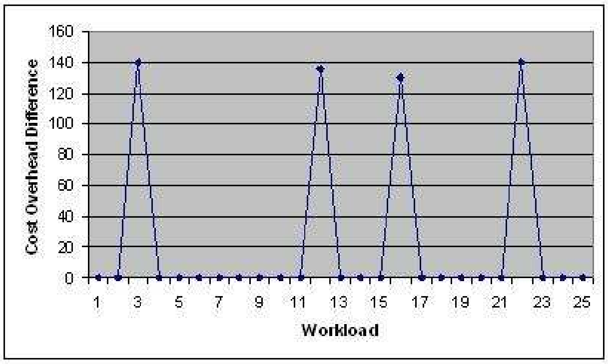

Figure 3: Comparison of Brute Force and Hill Climbing

< 0). This is because hill climbing does not try and encrypt attributes in any step after they have been encrypted once in the initial guess. Such a modification to the algorithm would probably return more accurate results.

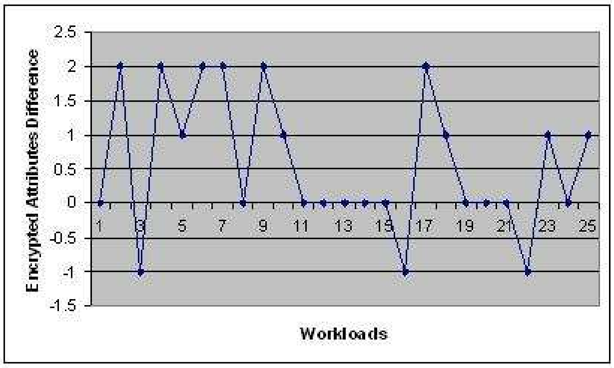

Figure 4: Number of Encrypted Attributes

7. RELATED WORK

Secure Database Services: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 [13]. Schemes proposed so far for this model encrypt data on the client side and then store the encrypted database on the server side [12, 11, 4]. 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 [17]. Our architecture of using multiple servers helps to achieve both efficiency and provable privacy together.

There is a wide consensus that privacy is a corporate responsibility [16]. In order to help and ensure corporations fulfil this responsibility, governments all over the world have passed multiple privacy acts and laws, for example, Gramm-Leach-Bliley (GLB)Act [8], Sarbanes-Oxley (SOX) Act [21], Health Insurance Portability and Accountability Act (HIPAA) [15] are some such well known U.S. privacy acts.

Similar ideas of fragmenting data for privacy have been proposed in [18].

8. CONCLUSIONS

The Hill Climbing does well overall and approximately 80% of the time returns a partition with cost overhead which matches the brute force approach.

Some improvements for Hill Climbing are:

- Execute hill climbing multiple times with different initial partitions and pick the best partition in the end. This will return better results.
- Use better heuristics of the initial partition like "the min cut" to compute the partition based on the affinity matrix as discussed in [2].
- Start with all attributes encrypted as the initial partition. Use the affinity matrix from [2] in the Weighted set cover to get a better initial guess.
- Encrypt attributes in the hill climbing process which might yield better end results.

Also, more work in the following areas would give useful results.

- Processing aggregate queries with GROUP BY and aggregate functions
- Perform similar analysis for semijoin plans and centralized plans which fetch data in more than one step from the servers.
- Support for more than one relation.
- Support for more than two servers.

9. REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

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

[7] 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.

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

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

[10] A. Gupta. Improved results for directed multicut. In *Proc. 14th Ann. ACM-SIAM SODA*, pages 454–455, 2003.

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

[12] 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.

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

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

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

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

[17] M. Kantarcioglu and C. Clifton. Security issues in querying encrypted data. Technical Report TR-04-013, Purdue University, 2004.

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

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

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

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