PSALM: Cardinality Estimation in the Presence of Fine-grained Access Controls

Huaxin Zhang, Ihab F. Ilyas, Kenneth Salem

University of Waterloo

h7zhang,ilyas,kmsalem@uwaterloo.ca

***Abstract*— In database systems that support fine-grained access controls, each user has access rights that determine which tuples are accessible and which are inaccessible. Queries are answered as if the inaccessible tuples are not present in the database. Thus, users with different access rights may get different answers to a given query. To process queries efficiently in the presence of fine-grained access controls, the database system needs accurate estimates of the number of tuples that are both accessible according to the access rights of the submitting user and relevant according to the selection predicates in the query.**

In this paper, we present PSALM, a sampling-based cardinality estimation technique for use in the presence of fine-grained access controls. Our technique exploits the fact that access rights are relatively static and are common to all queries that are evaluated on behalf of a particular user. We show that PSALM provides more accurate estimates than techniques that do not exploit knowledge of access rights.

I. Introduction

Access controls determine which data are accessible to each database user. Fine-grained access controls allow access rights to be specified on a per-tuple basis. Under the so-called *Truman model* [1], a user's queries are answered as if the database includes only those tuples that are accessible to the user. Variations of the Truman model are commonly used by systems that support fine-grained access controls, including Oracle [2], [3], DB2 [4], and SQL Server [5]. For example, Oracle Private Database customizes each user's query by extracting the user's *access control predicates* from the user's profile and appending the predicates to the *where* clause of the query [3]. The access control predicates filter tuples that are inaccessible to the user.

Suppose that a query includes a predicate $P_{Q(T)}$ on one of its input relations, T. $P_{Q(T)}$ may be a simple or complex predicate. Suppose further that a user's access rights for relation T are defined by an access control predicate $P_{AC(T)}$. Tuples are accessible to this user if and only if they satisfy $P_{AC(T)}$. For our purposes, the form of $P_{AC(T)}$ is not important. It may be an actual SQL predicate, as in Oracle VPD [3], or it may be implemented as an externally-defined function, as is the case in label-based access control systems [2], [4], [5]. For the purposes of our work, it is only necessary to be able to evaluate the access control predicate given a tuple from relation T.

The effect of the fine-grained access controls is to replace each $P_{Q(T_i)}$ in the user's query with $P_{Q(T_i)} \bigwedge P_{AC(T_i)}$, where T_i is a relation referred in the query. To create efficient query plans in the presence of fine-grained access controls, a query optimizer needs accurate cardinality estimates that account for the effect of the access controls. That is, instead of estimating the number of tuples that satisfy $P_{Q(T_i)}$, the optimizer must estimate the number of tuples that satisfy *both* the query predicate and the access controls. This paper addresses the problem of accurately estimating this cardinality. Because our technique is applied separately to each relation, in the remainder of the paper, we focus on one relation and use P_Q and P_{AC} rather than $P_{Q(T_i)}$ and $P_{AC(T_i)}$.

There are several ways to estimate the cardinality of conjunctive predicates like $P_Q \wedge P_{AC}$. For example, one can estimate the cardinality of of P_Q and P_{AC}, and then combine those estimates. To combine the estimates of the individual conjuncts, it may be necessary to make assumptions about the independence of the database attributes involved in the two predicates. Alternatively, the query optimizer may have information about correlations among attributes [6], or perhaps even a multidimensional histogram [7], [8], [9], [10] characterizing the joint distribution of all of the attributes involved in the predicates.

However, none of these techniques exploits an important characteristic of predicates like $P_Q \wedge P_{AC}$ that arise from fine-grained access controls: the access control predicate P_{AC} on a given table is the same for *all* queries issued by a particular user. A user's access control predicate is relatively static over time, as it changes only in response to changes in the system's access control policies. We expect that such changes occur much less frequently than queries. When fine-grained access controls are used, *every predicate in every query* will includes a relatively static component that arises from the access controls.

In this paper we consider cardinality estimation techniques that are able to exploit the static nature of P_{AC} in generating cardinality estimates for $P_Q \wedge P_{AC}$. The techniques that we consider are based on tuple sampling. Sampling-based techniques have several advantages for our application. Sampling works well even for high-dimensional predicates, i.e. predicates that involve many attributes. This is important because when access control predicates are applied to a base query, the number of attributes involved in the query predicates may increase. In addition, sampling based techniques can be applied easily even when the access control predicates are implemented as externally-defined functions that are black boxes from the perspective of the query optimizer.

The primary contribution of this paper is a sampling-based

TABLE I

SYMBOLS USED IN THIS PAPER

U	number of users in the system
P_Q	query selection predicate
P_{ACi}	access control predicate for the i^{th} user
N	cardinality of the target relation
N_i	number of tuples satisfying P_{ACi}
$N_{i,j}$	number of tuples satisfying both P_{ACi} and P_{ACj}
C_i	number of tuples satisfying $P_Q \wedge P_{ACi}$
$\tilde{C}_i$	estimate of C_i
n_i	number of tuples in i^{th} user's private sample
c_i	number of tuples in i^{th} user's private sample that satisfy $P_Q \wedge P_{ACi}$
ϵ_i	estimation error for i^{th} user
$\hat{\epsilon}_i$	estimation error bound for i^{th} user
$\hat{\epsilon}_{mean}$	mean estimation error bound over all users
Δ	confidence level for estimation error bounds

cardinality estimation technique called PSALM (Partitioned SAmpLing for Multiple users). PSALM leverages the fact the P_{AC} is fixed for each user and is relatively static. We show that cardinality estimates produced by PSALM are more accurate than those that can be produced using multi-dimensional histograms as well as those produced using simpler sampling-based approaches. We also show probabilistic bounds on the accuracy of the cardinality estimates produced by PSALM. In addition, we show how to extend PSALM to handle disjunctive access control predicates that arise when access privileges are inherited by users through role hierarchies or group hierarchies.

Although fine-grained access controls are our primary motivation for this work, it is worth noting that PSALM is applicable in other situations in which frequently-used, static query predicates arise. One example of this would be a publish/subscribe system where users express their general interests using filters [11]. These filters can be treated in much the same way as access control predicates.

The remainder of the paper is organized as follows. Section II presents some preliminaries and defines our terminology and notation. Section III presents two simple sampling algorithms, which we use as baselines, to estimate the cardinality of access control predicates. Section IV describes a general framework for sampling-based approaches to the cardinality estimation problem, and Section V presents PSALM, which is a particular realization of this framework. Section VI presents an evaluation of the accuracy of the estimates produced by PSALM, which we compare to multidimensional historgrams and to the simpler sampling techniques described in Section III. Section VII describes an extension of PSALM that handles disjunctive access control predicates that arise when access rights are inherited through role hierarchies or group hierarchies. In Sections VIII and IX we present related work, and conclude.

II. DEFINITIONS AND NOTATION

Our problem is to estimate the selectivity of query predicates in the presence of access control predicates. We focus on the problem of estimating the cardinality of the result of applying the query predicate, P_Q, to a single access-controlled relation. When there are multiple relations, our estimation techniques can be applied independently to each relation.

We use P_{ACi} to denote the access control predicate for the i^{th} user on the target relation. We use N to denote the cardinality of the target relation, and N_i to denote the number of tuples from the target relation that are accessible to the i^{th} user. That is, N_i is the number of target relation tuples for which P_{ACi} is satisfied. Finally, we use C_i to denote the number of tuples from the target relation that satisfy $P_Q \wedge P_{ACi}$. C_i is the cardinality we wish to estimate, for the i^{th} user and the query predicate P_Q. Table I summarizes our notation.

Given a fixed space budget for cardinality estimation, our goal is to design estimation techniques with small estimation error. We use $\tilde{C}_i$ to denote an estimate of the cardinality C_i. Following earlier work in this area [12], [13], we define ϵ_i, the estimation error for the i^{th} user, to be

$$\epsilon_i = \frac{|\tilde{C}_i - C_i|}{C_i}$$

This metric characterizes the estimation error relative to the actual cardinality of the query result. For example, $\epsilon_i = 0.3$ indicates that the estimate is $\pm 30\%$ of the actual cardinality. Unlike absolute error metrics such as $|\tilde{C}_i - C_i|$ or $|\tilde{C}_i - C_i|/N$, our relative error metric reflects the fact that the same cardinality estimation error may be more significant to the query optimizer when the true cardinality is small than when the true cardinality is large. For example, if $|\tilde{C}_i - C_i| = 10000$ and $\tilde{C}_i = 100000$, the estimation error may have little effect on the optimizer. However, if $|\tilde{C}_i - C_i| = 10000$ and $\tilde{C}_i = 100$, the optimizer may significantly underestimate the cost of a candidate query plan. One disadvantage of our metric is that estimation error explodes as $C_i \rightarrow 0$. Fortunately, this is not a serious issue. To avoid the blowup, we simply avoid scenarios in which C_i is extremely small. Note that our cardinality estimation techniques do not require any calculations of estimation error, so this problem does not affect their behavior. We perform these calculations only for comparing the estimation techniques.

III. SIMPLE APPROACHES

We begin by presenting two simple sampling-based estimation techniques. The first uses a single uniform random sample to generate a cardinality estimate for any user. That is, estimates for all users are based on the same sample from the target relation. We refer to this technique as the *single-sample* approach. The second technique partitions the available sampling space and uses a separate, private sample for each user. Each user's cardinality estimations are based on that user's private sample. We refer to this technique as the *sample-per-user* approach.

A. The Single-Sample Approach

We can estimate the cardinality of $P_Q \wedge P_{ACi}$, for any user and any query predicate using a single random sample of n

tuples from the target relation. Such a sample can be built for the target relation with relatively low cost, using various techniques [14]. The samples can be stored in the database catalog, and can be reused for different queries. If desired, such a sample can be incrementally maintained in the face of tuple insertions and deletions in the target relation [15]. Alternatively, the target relation can be periodically resampled as necessary to account for updates.

To obtain a cardinality estimate for P_Q for the i^{th} user, we evaluate $P_Q \wedge P_{ACi}$ for each sample tuple, and count the number of tuples for which the predicate is true. An unbiased cardinality estimate can then be obtained by

$$\tilde{C}_i = c_i \frac{N}{n}$$

where c_i is the number of sample tuples matching $P_Q \wedge P_{ACi}$. Using the Chernoff inequality, the estimation error ϵ_i of single-sample cardinality estimates can be bounded, with probability $(1 - \Delta)$, as follows:

$$\epsilon_i \leq \hat{\epsilon_i} = \sqrt{\frac{4N}{nC_i} \log \frac{2}{\Delta}} \tag{1}$$

Here, Δ is referred to as the confidence level of the error bound $\hat{\epsilon_i}$. Note that the estimation error bound is inversely related to C_i which is the number of tuples that satisfy $P_Q \wedge P_{ACi}$. Thus, as either the query predicate P_Q or the user's access controls P_{ACi} become more selective, the estimation error bound increases.

B. The Sample-Per-User Approach

Another way to estimate the cardinality of $P_Q \wedge P_{ACi}$ is to create and maintain a separate sample for each user. Each user's sample is of size $n_i = n/U$ so that the total space used for all users' samples fits within the space budget n.

The sample for the i^{th} user is a simple uniform random sample of the tuples *that are accessible to the i^{th} user*. As was the case for the single-sample approach, we can draw all U such random samples using a single pass over the target relation by maintaining U separate reservoir samples in parallel during a single scan of the target relation. Each tuple encountered in the scan is considered separately and independently for inclusion in the sample for each user.

To estimate the cardinality of P_Q for the i^{th} user, we evaluate P_Q for each tuple in the i^{th} user's sample, and count the number of tuples for which the predicate is true. All other users' samples are ignored. An unbiased cardinality estimate can then be obtained by

$$\tilde{C}_i = c_i \frac{N_i}{n_i}$$

where c_i is the number of sample tuples matching P_Q in the i^{th} user's sample. Notice that this estimator makes use of N_i, the total number of target relation tuples that are accessible to the i^{th} user. This value can be determined exactly for every user during the same scan that is used to draw the tuple samples from the target relation. We can bound, with confidence level Δ, the estimation error as follows:

$$\epsilon_i \leq \hat{\epsilon_i} = \sqrt{\frac{4N_i}{n_i C_i} \log \frac{2}{\Delta}} \tag{2}$$

In this case, the estimation error is inversely related to C_i/N_i, which is the *conditional selectivity* of the query predicate P_Q, given that a tuple is accessible to the i^{th} user. Hence, unlike the single-sample approach, the estimation error of the sample-per-user approach is independent of the selectivity of the users' access control predicates.

A comparison of Equations 1 and 2 shows that the i^{th} user will have a tighter error bound under the sample-per-user approach than under the single-sample approach if $n_i/N_i > n/N$ This says that the i^{th} user is better off with a smaller, private sample if its sampling rate among the i^{th} user's accessible tuples is greater than the single sample's sampling rate from the entire target relation. Since $n_i = n/U$, the i^{th} user will benefit from a private sample if

$$N_i < \frac{N}{U} \tag{3}$$

For example, imagine a system in which some portion of data are made accessible to a group of users, and the data are partitioned among the users by access controls as equality predicates on the users' IDs. These users will have non-overlapping accessible data and many of them will end up having accessible data of size less than $\frac{N}{U}$. In this case, the sample-per-user approach will be more beneficial to these users.

IV. A Generalized Approach

Our analysis of the simple sampling techniques in Section III indicates that in some circumstances it is better to make users share a sample for cardinality estimation, while in other circumstances it is better to dedicate a private sample to a user. This raises some general questions about sampling for cardinality estimation in the presence of access controls. First, under what conditions is it beneficial to dedicate a private sample to a user? Second, if a user is to share a sample, with which other users should the sample be shared?

In this section we present a general framework for sampling-based cardinality estimation, within which these questions can be answered. First, we partition the users into k ($1 \leq k \leq U$) groups, where U represents the number of users. We use $\{G_1, G_2, \ldots, G_k\}$ to denote these groups. Next, we partition the available total sampling space budget, n, to give a sampling space budget n_{G_j} for each group. The sampling space budget is divided such that the sum of the groups' sampling budgets is n. For each group G_j, we draw a simple random tuple sample of size n_{G_j} from the *union* of the sets of accessible tuples of each user in that group. Each group's sample is drawn independently of the samples for the remaining groups.

To estimate cardinality for User u_i in group G_j, we apply $P_Q \wedge P_{ACi}$ to the tuples in G_j's sample. If c_i is the number of sample tuples that satisfy $P_Q \wedge P_{ACi}$, we estimate the number

of tuples that satisfy $P_Q \wedge P_{ACi}$ as follows:

$$\tilde{C}_i = c_i \frac{N_{G_j}}{n_{G_j}} \tag{4}$$

where N_{G_j} denotes the cardinality of the union of the sets of accessible tuples of all users in G_j. The resulting estimates will give a Δ-confidence error bound of

$$\hat{\epsilon}_i = \sqrt{\frac{4N_{G_i}}{n_{G_i}C_i} \log \frac{2}{\Delta}} \tag{5}$$

for the i^{th} user.

Both of the simple techniques described in Section III are special cases of this general framework. The single-sample technique uses a single group that includes all of the users. The sample-per-user technique uses U groups, with one user per group. Our framework allows for many other grouping possibilities between these two extremes.

A. Sample Design

To apply the framework, we must decide how to partition the users into groups, and we must determine how much of the total sample space budget to allocate to each group. We call this the *sample design* problem for cardinality estimation in the presence of access controls.

Definition 4.1 (Sample Design Problem): Given the user access control predicates P_{ACi} and a total sample space budget of n tuples, partition the users into k groups $\{G_1, G_2, \ldots, G_k\}$ and choose a sample size n_{G_j} for each group such that $\sum_{1 \le j \le k} n_{G_j} = n$. Choose the groups and the sample sizes so that the average estimation error bound $\hat{\epsilon}_i$ over all users is minimized[1].

The sample design problem has two subproblems: grouping the users and allocating sample budget to each group. We begin by showing how to allocate the sample budget, given a particular partitioning of the users into groups.

Theorem 4.1: If the query predicate P_Q is independent of the access control predicates P_{ACi} of all users, an optimal allocation $n_{G_j}, 1 \le j \le k$ among a given set of user groups under a total sample budget n is given by the following:

$$n_{G_j} = \frac{n \cdot H_j^{\frac{1}{3}}}{\sum_{j=1}^{k} H_j^{\frac{1}{3}}}$$

where H_j is $\sum_{(i:u_i \in G_j)} \frac{N_{G_j}}{N_i}$.

Proof: The total estimation error bound of all users can be rewritten as follows according to Equation 5:

$$\sum_{1 \le i \le U} \sqrt{\frac{4 \log \frac{2}{\Delta} \cdot N_{G_i}}{n_{G_i} \cdot N_i} \cdot \frac{N_i}{C_i}}$$

Here, $\frac{C_i}{N_i}$ falls within 0 and 1 for all access controlled query from User u_i. We use $Pr_i(x)$ to represent the probability density function for $\frac{C_i}{N_i}$ in its domain $(0, 1]$[2]. The *expected total estimation error bound* for all queries for all users is thus:

$$\sum_{1 \le i \le U} \sqrt{\frac{4 \log \frac{2}{\Delta} \cdot N_{G_i}}{n_{G_i} \cdot N_i}} \int_0^1 \sqrt{\frac{1}{x_i}}\, Pr_i(x_i)\, dx_i$$

Because we have no prior knowledge on the workload among all users, one reasonable approach is to assume independence between P_Q and P_{AC}. Therefore, the integration part of the above will be a constant ct for all users. and we can rewrite the expected total estimation error as follows (we remove ct from the expression for simplicity):

$$\sum_{1 \le j \le k} \sqrt{\frac{4 \log \frac{2}{\Delta} \cdot N_{G_j}}{n_{G_j}} \cdot \sum_{(i:u_i \in G_j)} \frac{1}{N_i}}$$

If we have chosen a user grouping (and thus a fixed $\mathcal{G}$), we can rewrite the above as

$$\sum_{j=1}^{k} \sqrt{\frac{4 \log \frac{2}{\Delta}}{n_{G_j}} \cdot H_j}$$

where H_j is $\sum_{(i:u_i \in G_j)} \frac{N_{G_j}}{N_i}$ and is fixed value for a chosen user-grouping.

Under the constraint $\sum_{j=1}^{k} n_{G_j} - n = 0$, the Lagrange formula for the minimum average estimation error bound becomes:

$$\sum_{j=1}^{k} \sqrt{\frac{4 \log \frac{2}{\Delta}}{n_{G_j}} \cdot H_j} + \lambda \left(\sum_{j=1}^{k} n_{G_j} - n \right)$$

We wish to find a configuration that gives a zero gradient for λ and each n_{G_j} in the above formula, which gives the result in the theorem.

■

From Equation 5, we can determine that the mean estimation error bound under this optimal sample space allocation from Theorem 4.1 will be given by

$$\epsilon = \sqrt{4 \log \frac{2}{\Delta}} \cdot n^{\frac{1}{2}} \cdot \left(\sum_{j=1}^{k} H_j^{\frac{1}{3}} \right)^{\frac{3}{2}} \tag{6}$$

This is the minimum estimation error bound under the assumption (from Theorem 4.1) that the query predicate is independent of the access control predicates. It is much more difficult to determine an optimal allocation and corresponding error bound if this is not the case. However, it is important to note that this independence assumption is made *only* for

[1]Instead of *average estimation error bound*, other metrics like *maximum estimation error bound* can be tackled in a similar fashion if desired, by adjusting the formulae in this paper.

[2]the integration domain start from somewhere bigger than zero to avoid division by zero

the purpose of sample space allocation. The actual estimation of cardinalities using those samples does not rely on the independence of query predicates and access controls.

The sample design problem is now reduced to the problem of finding an optimal grouping of users. The number of possible groupings of U users is the same as the number of partitions of U different items, which is known as the Bell Number [16]. If we use B_U to denote the Bell Number of U items, we have the following recursive representation for B_U:

$$B_{U+1} = \sum_{k=0}^{U} \left(\begin{array}{c} U \\ k \end{array} \right) B_k$$

which has a non-recursive representation as Dobinski's Formula [16]:

$$B_U = \frac{1}{e} \sum_{k=0}^{\infty} \frac{k^U}{k!}$$

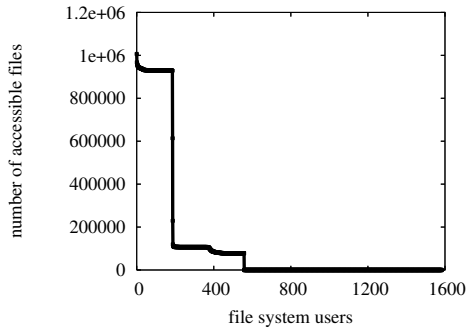
Since the number of possible user partitions B_U is exponential in the number of users, a brute force enumeration of all possible user-groupings is infeasible. Whether there exists a polynomial time algorithm to identify the optimal partition is still unknown. However, in Section V we introduce a polynomial time approach which uses heuristics to aggressively prune the search space. Our proposed technique is not guaranteed to choose optimal user groups. However, it is guaranteed to perform at least as well as the simple techniques described in Section III.

V. PSALM

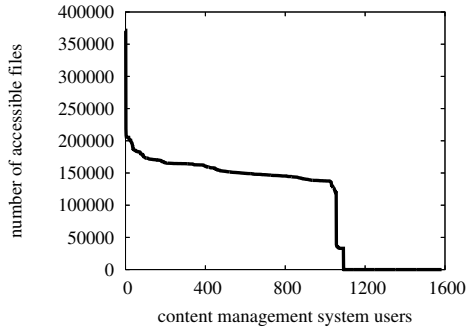
PSALM is a realization of the generalized sampling framework that was presented in Section IV. Users are partitioned into groups, and a sample is created for each group. Cardinality estimates are generated using Equation 4, and they have error bounds given by Equation 5.

To partition the users, PSALM is guided by two observations. The first concerns *access privilege skew*. Systems that provide access controls may include a mix of high-privileged users, who have access to most of the data, and low-privileged users, who have very limited access. For example, in previous work [17], [18] we studied access controls in several different systems, including a general purpose university file system and a content management system. Figure 1(a) shows the number of files that are accessible to different users in the file system that we studied. There are two obvious groups of users in the system, with the high-privilege group having access to almost ten times as many files as the low-privilege group. Figure 1(b) shows a similar graph for the content management system. Here, the picture is more complex, as there appear to be three distinct types of users in terms of the number of files that are accessible. However, as was the case for the file system, the distribution exhibits a significant amount of skew.

The second observation is that there may be strong correlations among the access controls of different users [17]. For example, users that are involved in common activities are likely to share access to data related to those activities. By identifying and measuring these correlations, we can use them to guide user grouping.



(a) university file system



(b) content management system

Fig. 1. Data Accessibility for Different Users in Two Systems

PSALM_DESIGN_PROC()

PHASE ONE

partition users into two groups, $\mathcal{U}_H$ and $\mathcal{U}_L$:
$\mathcal{U}_H$ includes high-privilege users
$\mathcal{U}_L$ includes low-privilege users
choose a sample size n_0 for $\mathcal{U}_H$
choose a sample size n_i for each user $u_i \in \mathcal{U}_L$

PHASE TWO

partition the users in $\mathcal{U}_L$ into groups based on access rights correlations
set the sample size of each group to be the sum of the sample sizes of the group's members

Fig. 2. Sample Design in PSALM

PSALM uses a two-phase approach, based on these observations, to arrive at a sample design. Figure 2 gives a high-level overview of PSALM's approach to sample design. In Sections V-A and V-B we describe the two phases in more detail.

A. Phase One: Access Privilege Skew

High-privilege users can profitably share a single sample for cardinality estimation, since such users will share access to many of the same tuples. On the other hand, low-privileged users are better off with private samples of their accessible

PHASE_ONE_USER_PARTITION()

1: Sort the users in ascending order of number of accessible tuples.
Let u_i represent the ith user in this sorted order, and N_i represent the number of tuples u_i can access.
2: Find p from $\{0, \ldots, U\}$ such that
$p + \left(\sum_{i>p} \sqrt{\frac{N}{N_i}} \right)^{\frac{2}{3}}$ is minimized
3: Set $\mathcal{U}_H = \{u_i : i > p\}$
$\mathcal{U}_L = \{u_i : i \leq p\}$
4: Return $\{\mathcal{U}_L, \mathcal{U}_H\}$

Fig. 3. Identifying $\mathcal{U}_H$ and $\mathcal{U}_L$

tuples. To account for this, PSALM separates users into two categories: high-privilege users and low-privilege users. We use $\mathcal{U}_H$ to denote the set of high-privilege users and $\mathcal{U}_L$ to denote the set of low-privilege users. PSALM chooses $\mathcal{U}_H$ and $\mathcal{U}_L$ so that the mean estimation error bound $\hat{\epsilon}_{mean}$ over all users is minimized. The mean estimation error bound is calculated as if sampling were done using the generalized sampling framework, with

> one group consisting of all users in $\mathcal{U}_H$,
> one group for each user in $\mathcal{U}_L$, and
> sample sizes for each group chosen according to Theorem 4.1.

In phase one, PSALM determines which users to place into $\mathcal{U}_H$ and which to place into $\mathcal{U}_L$ so that the estimation error bound will be minimized. In order to do this, PSALM uses the algorithm shown in Figure 3. Under this algorithm, PSALM considers $U + 1$ possible sets of users to include in $\mathcal{U}_H$: the empty set, the set consisting of the highest privilege user, the set consisting of the two highest privilege users, and so on. Theorem 5.1 indicates that the optimal $\mathcal{U}_H$ will be among those considered by the algorithm of Figure 3.

Theorem 5.1: The partition ($\mathcal{U}_H$, $\mathcal{U}_L$) returned by Algorithm 3 minimizes the mean estimation error bound $\hat{\epsilon}_{mean}$ over all possible two-way partitions of the users.

Since both simple cardinality estimation techniques presented in Section III are special cases of PSALM, we have the following corollary to Theorem 5.1:

Corollary 1: Cardinality estimation using the $\mathcal{U}_H$ and $\mathcal{U}_L$ produced by Algorithm 3 results in a mean estimation error bound at least as low as the error bounds achieved by the single-sample technique and the sample-per-user technique.

Thus, the sample design produced by phase one of PSALM is sufficient to ensure that PSALM will be at least as accurate as the simple techniques. By considering access privilege correlation in phase two, PSALM seeks to further improve its accuracy by refining the sample design from phase one.

B. Exploiting Access Privilege Correlation

The first phase of the PSALM algorithm paritions the users into $\mathcal{U}_H$ and $\mathcal{U}_L$. The second phase, described in this section, combines low-privilege users from $\mathcal{U}_L$ into groups based on correlations among the users' access rights. Grouping low-privileged users further reduces the mean estimation error bound of PSALM sampling. In the remainder of this discussion, we assume that the first phase of PSALM has partitioned the users into $\mathcal{U}_L$ and $\mathcal{U}_H$, and has selected a shared sample size of n_0 for users in $\mathcal{U}_H$ and a same sample size n_L for each of the users in $\mathcal{U}_L$ according to Theorem 4.1. Let $G \subseteq \mathcal{U}_L$ be a group of low-privileged users. Instead of creating a separate, private sample for each user in G, PSALM uses a single, combined sample of size $|G|n_L$ that is shared by all users in G. This shared sample is a simple random sample drawn from among all tuples that satisfy P_{ACG}, defined as follows:

$$P_{ACG} = \bigvee_{i \in G} P_{ACi}$$

That is, we sample from among those tuples that are accessible to at least one user in the group. To compute an unbiased cardinality estimate for predicate P_Q for User $u_i \in G$, we use Equation 4. The resulting cardinality estimates will have Δ-error bounds given by Equation 5.

Grouping users in this way does not change the cardinality estimates of users outside of G, nor does it affect the accuracy of those estimates. Thus, any change in the overall average estimation error bound that results from this grouping will come from the change in the estimation error bound of the users in G. By combining Equations 2 and Equation 5, user grouping will result in a lower mean estimation error bound among the users in the group if the following inequality holds:

$$\sum_{i \in G} \sqrt{\frac{N_G}{|G|C_i}} < \sum_{i \in G} \sqrt{\frac{N_i}{C_i}}$$

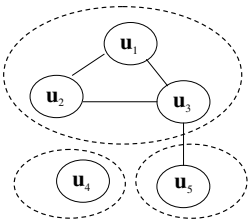
When there is little overlap among the accessible tuples of the grouped users, N_G approaches $\sum_{i \in G} N_i$ and there is little benefit to grouping users. However, if the grouped users have many accessible tuples in common, then N_G is much smaller and grouping will result in improved estimates for all users in the group because of the larger size of the group's sample. Thus, PSALM identifies users whose access rights are correlated, and group them together.

To determine which users to group together, we first define a pairwise similarity function, SIM, over the users in $\mathcal{U}_L$. The similarity between the i^{th} and j^{th} users is defined as follows:

$$SIM(i,j) = \min\left(\frac{N_{i,j}}{N_i}, \frac{N_{i,j}}{N_j}\right) \quad (7)$$

Here, $N_{i,j}$ is defined as the number of tuples from the target relation that are accessible to both User i and User j. This similarity function has properties similar to other set similarity functions like the Jaccard Coefficient [19], i.e., it has a value between 0 and 1, and a value close to 1 indicates a strong similarity, while a value of 0 indicates no similarity.

	u_1	u_2	u_3	u_4	u_5
t_1	1	1	1	0	0
t_2	1	1	1	0	0
t_3	0	0	1	0	1
t_4	0	0	1	1	1



Min cluster partition with AC similarity thresh-hold 0.45

Fig. 4. An example of user grouping

Using function SIM, we can define a *similarity graph* as an undirected graph with one node for each user in $\mathcal{U}_L$. There is an edge between User i and User j if and only if $SIM(i,j) \geq \theta$, where θ is a tunable parameter of the grouping algorithm. We will describe how to pick θ in Section V-B.1 and in Section VI-C.

To place the users into groups, we attempt to find a *minimum clique partition* [20] of the user similarity graph. This identifies a set of non-overlapping cliques that, together, cover the entire user similarity graph. Each such clique becomes one of the user groups for which we create a sample. By using cliques as sampling groups, we ensure that all users in a group have pairwise-similar access rights. By minimizing the number of cliques, we minimize the number of separate samples that are required, thus allowing us to use larger samples while remaining within the space budget.

Figure 4 illustrates the process of identifying user groups through a scenario in which there are five users $(u_1, u_2, \ldots, u_5)$ and four tuples $(t_1, t_2, \ldots, t_4)$. The matrix indicates which tuples are accessible to each user, with a "1" and "0" indicating accessibility and inaccessibility respectively. Using a similarity threshold θ of 0.45, we obtain the user similarity graph with pairwise connections among u_1, u_2 and u_3, as shown in Figure 4. A minimum clique partition for this graph is illustrated using dashed lines.

The problem of finding a minimum clique partition is NP-complete [20]. Thus, we use a randomized greedy algorithm to partition the graph. The algorithm produces a set of non-overlapping cliques that cover the user similarity graph. However, the set is not guaranteed to have minimum cardinality. The greedy algorithm first finds a maximal clique in the graph by starting from a random node, finding the clique around the node, and removing all nodes in the clique and all edges induced by these nodes. This process continues until there are no more nodes left. We repeat this greedy algorithm several times from randomly-selected initial nodes and record the smallest clique partition that is found.

1) Refining User Groups: Although grouping correlated users reduces the mean estimation error bound of those users, some individual users in the group may incur a loss in estimation accuracy. For example, consider two users u_i and u_j from $\mathcal{U}_L$. Suppose each user would have a private sample size of n_L tuples without correlation-based grouping. If u_i and u_j are grouped based on correlation, they have a total sample size of $2n_L$ tuples. If we randomly sample $2n_L$ tuples from the union of their sets of accessible tuples, User u_i will have the following expected number of accessible tuples included in the sample:

$$2n_L \cdot \frac{N_i}{N_i + N_j - N_{i,j}}$$

If N_j is much larger than N_i, then according to the above formula, the expected number of sample tuples for u_i will be less than his original sample quota n_L. This means that u_i may have a higher estimation error bound after grouping with u_j. The following lemma describes the conditions under which users u_i and u_j can be grouped together so that both of them will have a lower estimation error bound:

Lemma 5.1: Both User u_i and User u_j will have lower estimation error bounds after grouping if $N_{i,j} > |N_i - N_j|$, where $N_{i,j}$ denotes the number of tuples that are accessible to both User u_i and User u_j.

Lemma 5.1 can be extended to group more than two users. The following theorem is based on the access control similarity threshhold θ computed from Formula 7.

Theorem 5.2: Suppose users $\{u_1, u_2, \ldots, u_{|G|}\}$ have access controls with pairwise similarity above θ as defined in Formula 7. All of these users will have a lower cardinality estimation error bound after merging all their sample quotas if the following holds for every user $u_i, 1 \leq i \leq |G|$:

$$\theta > \frac{\sum_{j=1}^{|G|} N_j - |G| \cdot N_i}{\sum_{j=1}^{|G|} N_j - N_i} \tag{8}$$

To ensure that no user will have a higher estimation error bound after grouping, we need to verify that Equation 8 is satisfied for every user in each clique when performing the greedy clique-partitioning algorithm as described in Section V-B. If this formula is not satisfied for some users in a clique, we remove these users from that clique. We keep removing users until this formula is satisfied for every user in each clique.

VI. Evaluation

We use synthetic databases and synthetic workloads to evaluate the quality of the cardinality estimates produced by PSALM. We compare PSALM's estimates to those produced by the two simple techniques described in Section III, and also to estimates obtained using multidimensional histograms. We next describe how we generated the database and workloads, and present the results of our evaluation.

A. Simulation Setup

Our experiments use synthetic data created using an approach similar to that of Bruno, Chaudhuri and Gravano in their evaluation of STHoles multidimensional histograms [8]. We first create 500,000 tuples from a data domain of $[0, 1000)^d$, where d_D is a given database dimensionality, i.e., the number of attributes in each tuple. We experimente with two-dimensional, four-dimensional, and eight-dimensional tuples. These tuples are distributed among 100 clusters in the

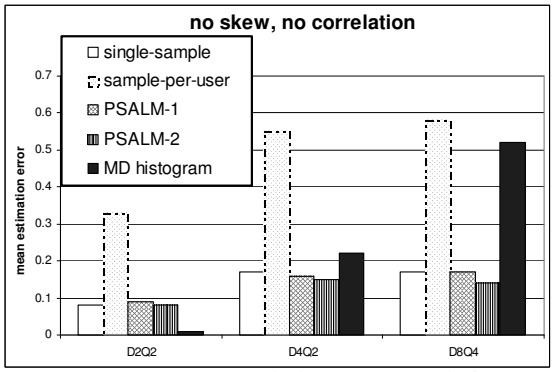
data domain. We first create 100 evenly distributed tuples as the center (mean) of each cluster, then, for each cluster, we randomly create tuples following a Gaussian distribution with its mean at the cluster center and a standard deviation of 25. The number of tuples in each cluster is chosen using a Zipfian distribution with parameter 1.0.

The database workload consists of multidimensional range queries generated using an approach similar to that of Pagel et al. [21]. Query predicates are conjunctive range predicates in d_Q dimensions, where d_Q is a given query dimensionality. To generate a query, we choose d_Q dimensions randomly and, for each dimension, choose a random center for that dimension's range. The width of the range is fixed to be 100 times the number of conjunctive range predicates in the query. This way the query has non-zero selectivity even if the number of conjuncts goes up. We choose the range centers in one of three ways: type A ranges have their centers chosen from the center of the clusters in the data, type B ranges have their centers following a uniform distribution in the data domain, and type C ranges have their query centers chosen to be independent of the data distribution and following a Gaussian distribution. Similar predicates have been used by Bruno, Chaudhuri and Gravano [8]. All of the experiments reported here used type B for query predicates and type A for access control predicates (described next). Other combinations that we have tested result in trends similar to the ones reported here.

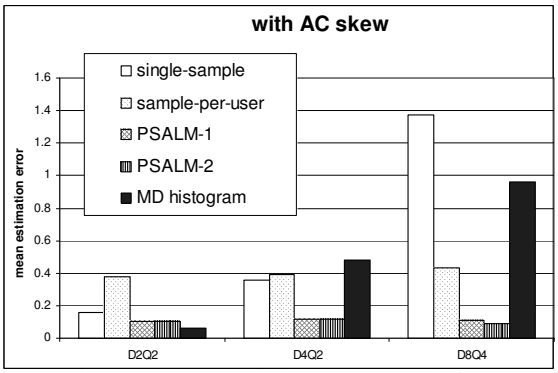
Our experiments simulate 100 users. For each user, we create a randomly generated access control predicate $P_{AC.}$ Like query predicates, access control predicates are multi-dimensional conjunctive range predicates in d_Q dimensions, and are generated in the same way as the query predicates. The dimensions of the access control predicates are randomly chosen to be independent of the query predicates. We run experiments with access privilege skew and without access privilege skew. To model access privilege skew, we consider half of the users to be high-privilege users, and the other half to be low-privilege users. We give high-privilege users more access rights by using larger access control predicate ranges for those users than for low-privilege users. For high-privilege users, we use an access control range predicate width of 300 in each predicate dimension. For low-privilege users, we use a predicate width of 80. For the experiments without access privilege skew, access control predicates for all users have a width of 100 in each dimension. To simulate access correlation, we generate similar access control predicates among the low-privileged users. This is done by creating access control range predicates from the same Gaussian distribution for all of the low privileged users.

We implemented both the single-sample technique and the sample-per-user technique described in Section III. We implemented two versions of PSALM, which we call PSALM-1 and PSALM-2. PSALM-1 includes the first phase of the PSALM algorithm, but not the second. That is, high-privilege users share a single sample, but there is no attempt to group low-privilege users based on access right correlation. PSALM-2 is the full PSALM algorithm, including both phases. For the test data we used, it takes on average 5 seconds to build PSALM-1 samples and less than 1 minute to build PSALM-2 samples. Finally, we implemented multidimensional histograms. Our histograms use an equi-width grid-based implementation similar to that of Aboulnaga and Chaudhuri [7]. All of these techniques were constrained to operate under the same space budget which, in all of our experiments, was sufficient to hold 1000 d_D-dimensional tuples.

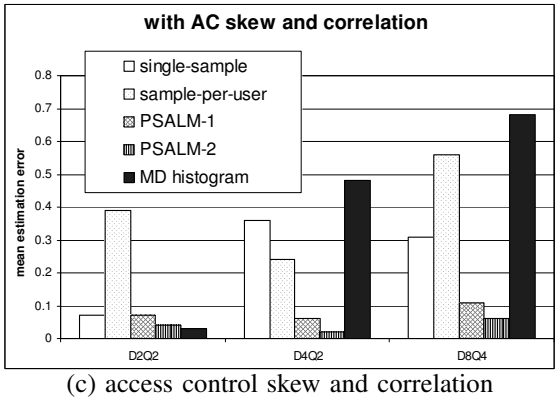
B. Experiments



(a) no access control skew, no correlation



(b) access control skew, no correlation



(c) access control skew and correlation

Fig. 5. Accuracy of Cardinality Estimates

Each of our experiments is characterized by the values of parameters d_D and d_Q. We use "Dd_DQd_Q" to identify a particular experiment. For example, $D4Q2$ denotes and experiment with a 4 dimensional database and 2 dimensional

query predicates and user access control predicates. First, we generate 100 access control predicates representing the 100 users, and then, we generate 10 query predicates, and evaluate all cardinality estimation techniques for each query and for each user. For each experiment, we measure the actual cardinality of the result and calculated the estimate cardinality using each of the cardinality estimation techniques. For each estimation technique, we report the average estimation error over all queries.

Figure 5 shows a comparison of the mean estimation error of the cardinality estimation techniques that we considered. The figure shows results from 3 access control scenarios, one in which there is neither access control skew nor access control correlations among the low-privilege users (Figure 5(a)), one in which there is skew but no correlations (Figure 5(b)), and one with both access control skew and correlations (Figure 5(c)). For each scenario, we report results for D2Q2, D4Q2, and D8Q4.

In the two-dimensional case (D2Q2), multidimensional histograms gives the most accurate estimates. However, PSALM-2 performs well, and its accuracy approaches that of the multidimensional histogram in the scenarios with access rights skew and correlation. In the four- and eight-dimensional cases (D4Q2 and D8Q4), the accuracy of multidimensional histograms deteriorate quickly, while PSALM was able to maintain its accuracy.

When there is no access privilege skew (Figure 5(a)), the simple single-sampling technique performs approximately as well as PSALM. However, PSALM's performance is significantly better than that of single-sampling when skew is introduced (Figure 5(b) and Figure 5(c)). In these scenarios, the high-privilege users have, on average, access to about 10 times as many tuples as the low-privilege users. PSALM chooses to use a single sample for the high-privilege users and separate samples for each low-privilege users.

A comparison of the accuracy of PSALM-1 and PSALM-2 in Figure 5(c) shows that the second phase of PSALM is beneficial when there are access privilege correlations among the users. In each case (D2Q2, D4Q2, and D8Q4), PSALM-2's estimation error is about half that of PSALM-1, which does not exploit correlation.

C. Tuning the Similarity Threshold

One important question is how to determine the similarity thresholds θ when grouping users with correlated access controls. The higher the threshold, the fewer users can be grouped; the lower the threshold, the smaller accuracy improvement from each group. We experimented on access control data from an OpenText LiveLink[3] content management system. This system has approximately $370,000$ objects and 1584 users. We randomly select sets of users from among all 1584 users, and we apply our user grouping algorithm in Section V-B to each selected set of users, with different θ ranging from 0.3 to 0.7.

[3]LiveLink is a trademark of OpenText Corporation.

Figure 6 shows the number of user-groups after grouping, with different threshholds θ. We find that the user grouping algorithm effectively reduces the number of samples. However, the effect of θ is not that significant. The reason is that while a lower θ value introduces larger cliques and results in fewer groups, many users in the cliques do not satisfy Formula 8, and have to be discarded.

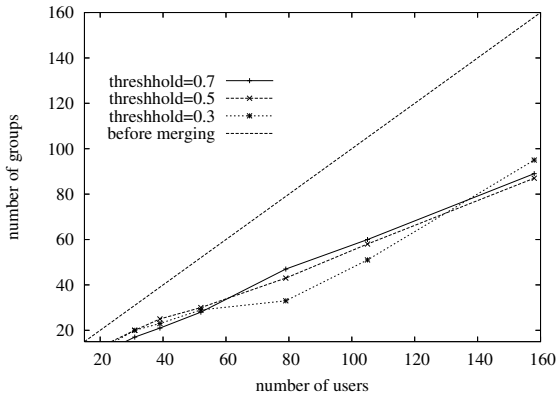
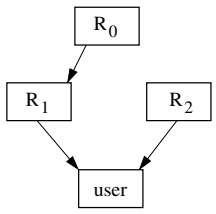


Fig. 6. Effect of grouping on the number of samples required for low-privilege users

VII. Disjunctive Access Controls

Fine-grained access controls are not always explicitly specified for each user in the system. For example, in role-based access control systems, an administrator defines several roles and assigns access rights to these roles. Roles, in turn, are assigned to users, in which case all access rights of the role are granted to the user. Roles can also be assigned to other roles. This defines a hierarchy of users and roles, as illustrated in Figure 7, in which each user obtains access rights from all the roles to which he is directly or indirectly assigned. In other words, the set of tuples accessible to a user is the *union* of the sets of tuples that are accessible to that user's ancestors in the role hierarchy. For example, the user in Figure 7 obtains access rights from three roles R_0, R_1, R_2. A similar situation arises in systems that allow a user group hierarchy to be defined, and that support the inheritance of group access rights by members of the group.



tuples accessible to roles:
$R_0 : t_1, t_4, t_6$
$R_1 : t_2, t_4, t_5$
$R_2 : t_1, t_3, t_4$
tuples accessible to user:
$t_1, t_2, t_3, t_4, t_5, t_6$

Fig. 7. A Role Hierarchy

Suppose a user has been assigned k roles, either directly or indirectly, and that each role R_i has an access control predicate P_{ACi} on a particular target relation. The user's access control predicate on the target relation is thus $(\bigvee_{i\in\{1,\ldots,k\}} P_{ACi})$. Given query selection predicate P_Q, we need to estimate

the cardinality of the set of target relation tuples satisfying $P_Q \wedge (\bigvee_{i\in\{1,\ldots,k\}} AC_i)$.

A straightforward approach to compute the estimated cardinality is to compute the *effective access controls* for each user, based on his role assignments, and then use PSALM to estimate cardinalities. However, the effective access controls for all users may be costly to compute. Moreover, we ignore the disjunctive structure of these predicates, and may require to create one PSALM samples for each user in the worst case.

An alternative is to record an access control predicate for each *role*, and to compute cardinality estimates for these individual role predicates using PSALM. To estimate the cardinality for a particular user's access control queries, we combine the PSALM-generated estimates for that user's roles to produce a cardinality estimate that reflects the disjunction of role predicates for the particular combination of roles to which that user is assigned. The primary advantage of this approach is that the number of roles may be much smaller than the number of users. PSALM's sample-based cardinality estimates for a few roles can be more accurate than its estimates for many individual users would be, since it will be able to devote more of its fixed sample budget for to each estimate. This approach is also appealing since it recognizes and exploits the disjunctive predicate structure that is induced by the role (or group) hierarchy.

To use this alternative approach, we must have means of accurately estimating the cardinality of a disjunctive access control predicate, given PSALM-produced estimates of the cardinality of the individual disjuncts. We begin by applying the distributive law to rewrite the access control predicate:

$$P_Q \wedge (\bigvee_{i\in\{1,\ldots,k\}} P_{ACi}) \equiv \bigvee_{i\in\{1,\ldots,k\}} (P_Q \wedge P_{ACi}) \quad (9)$$

We use PSALM to accurately estimate the cardinality of each $P_Q \wedge P_{ACi}(I)$. A variety of techniques can then be used to produce a cardinality estimate for the disjunction. In practice, these often rely on *ad hoc* formulas or independence assumptions, which can reduce accuracy. We propose, instead, to use the Coverage Algorithm [22] to estimate the cardinality of the disjunctive predicate. This requires that the cardinality estimate for each individual disjunct be determined using sampling. However, since we are using PSALM to generate those estimates, that is exactly what we have.

A. The Coverage Algorithm

The Coverage Algorithm's time complexity is polynomial in the number of disjuncts. The algorithm can be used to accurately estimate the size of the union $\bigcup_{i\in(1,\ldots,k)} C_i$, provided that [22]:

1) we can accurately estimate the size of each C_i, and
2) we can uniformly sample from each C_i, and
3) we can determine in polynomial time whether a given tuple belongs to C_i.

The Coverage Algorithm takes as input the sets $\mathcal{S} = \{C_i, i \in \{1,\ldots,k\}\}$ and proceeds as in Algorithm 8. The

COVERAGE$(\mathcal{S}, n)$

```
1: set counter W = 0;
2: sort sets in S by their sizes in descending order
3: for each C_i ∈ S
4:         sample n|C_i| / Σ_{C_k∈S} |C_k| tuples uniformly from C_i
5:         for each tuple t sampled
6:                 if t ∉ C_j, j ∈ {1,...,i−1} W++;
7: return W;
```

Fig. 8. Coverage Algorithm [22]

algorithm starts with a counter W with value zero, and a total sample size n. It then samples from each of the sets C_i, where the number of samples taken from C_i is proportional to $|C_i|$. For each tuple sampled from C_i, it determines whether the tuple belongs to any C_j such that $j \in \{1,\ldots,i-1\}$. If it does *not* belong to any of these sets, we increment the counter W by one.

After all of the samples from $C_1, C_2 \ldots, C_k$ have been checked W is used to compute the estimated cardinality $|C| = |\bigcup_{i\in(1,\ldots,k)} C_i|$ as follows:

$$|C| = \sum_{i\in\{1,\ldots,k\}} |C_i| \frac{W}{n} \quad (10)$$

It is possible to bound the accuracy of the cardinality estimates generated by the Coverage Algorithm, as described by Theorem 7.1.

Theorem 7.1 (Coverage Algorithm Accuracy [22]): The sampling procedure in Algorithm 8 and Equation 10 yield an ϵ-approximation to $|C|$ with $(1-\Delta)$ probability, provided sample size $n \geq \frac{4k}{\epsilon^2} \ln \frac{2}{\Delta}$.

B. Applying the Coverage Algorithm

We now show how to apply the Coverage Algorithm for estimating the cardinality of disjunctive predicates of the form shown in Formula 9. Given a target relation T, we need to be able to do the following to apply the Coverage Algorithm:

1) estimate the cardinality of each $P_Q \wedge P_{ACi}(T)$, and
2) sample uniformly from each $P_Q \wedge P_{ACi}(T)$, and
3) decide whether a given tuple satisfies $P_Q \wedge P_{ACi}$ in polynomial time.

The first requirement is met because we are using $PSALM$ to estimate those cardinalities. The second requirement can be satisfied by sampling from the samples that is produced by PSALM for estimating the cardinalities of the disjuncts. Specifically, to sample from $P_Q \wedge P_{ACi}$, we sample from among those tuples in PSALM's sample that satisfy $P_Q \wedge P_{ACi}$. The third requirement depends on the complexity of the access control predicates associated with each individual role.

C. Evaluating the Coverage Algorithm

We evaluated the Coverage Algorithm on top of the PSALM technique using a data set that describes American household expenditures [4]. This data set consists of 127931 tuples, with each tuple representing a family's expenses on insurance, property tax, electricity, gas, water, and fuel. There is some correlation among the six attributes.

We simulate users' access controls by randomly creating four roles. The access control of each role is a single predicate P_{AC} on one of the six attributes of the data. We then randomly assign these four roles to users. Each user may have one to four roles, and the user's access control predicate is a disjunct over the single attribute predicates from his roles. Therefore, there are altogether 15 distinct access controls from all our users. All of the disjunctions that we considered in this experiment had an actual selectivity of approximately 0.25, meaning that each user has access to approximately one quarter of the tuples. The reason for this selectivity will be clear later.

Our workload for this experiment consists of one-dimensional range queries with the same selectivity (0.25) as the access controls. We generated a series of such queries and, for each pair of query and user's access control predicate, calculated both the actual result cardinality and the estimated cardinality. We compared two approaches for cardinality estimation. The first is the PSALM combined with the Coverage Algorithm, as described previously. This approach uses PSALM to generate cardinality estimates for each of the four roles, and then combines those estimates using the coverage algorithm to obtain cardinality estimates for user queries. The second approach is to ignore the disjunctive structure of the access controls. Instead we treat each user's disjunctive predicate as an atomic predicate, and we apply PSALM directly on these 15 users to estimate the cardinality of that predicate.

Figure 9 shows the estimation error of the two estimation techniques, assuming that both techniques are given a sample space budget of 60 tuples. Each point in the figure represents a query from a user. The x-coordinate denotes the estimation error using PSALM directly on all users and the y-coordinate denotes the estimation error using the Coverage Algorithm on top of PSALM on the roles. In most cases, the estimates produced by the Coverage Algorithm are significantly more accurate than those produced by PSALM directly on users.

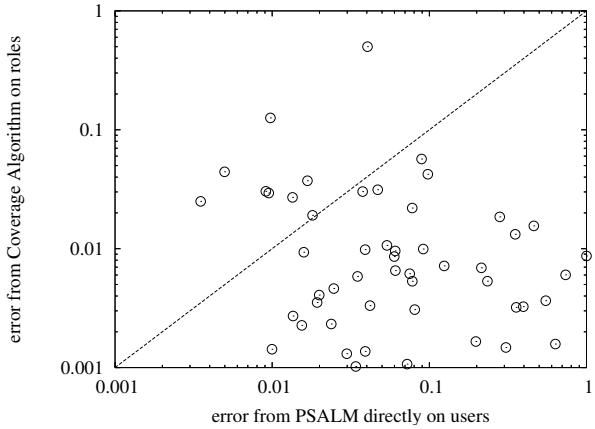


Fig. 9. Accuracy of Two Techniques for Estimating the Cardinality of Disjunctive Predicates

VIII. Related Work

There exists a broad literature on cardinality estimation techniques for conjunctive predicates without the attribute value independence assumption. Among them, multi-dimensional histograms [7], [8], [9], [10], wavelet-based techniques [23] and sampling [24], [25], [26] are the most common. We focus on sampling-based techniques in this paper.

There are two main approaches to sampling data for cardinality estimation. One is to sample for each query at runtime, either proactively before executing the query [27], or reactively after executing the query [28]. The other approach is to sample off-line without prior knowledge of the queries. Although PSALM takes advantage of prior knowledge access controls, it is closer to the off-line sampling category since it does not make any assumptions about the users' queries.

Kolmogorov's statistics show that a moderately-sized sample gives accurate selectivity estimation for queries, and that the required sample size does not depend on the size of the underlying dataset [29]. However, that evaluation is based on the relative error of selectivity, rather than the relative error of cardinality. It is well-known that uniform random sampling does not provide accurate cardinality estimation when the data distribution is highly skewed or the query results are very small. Therefore, instead of specifying a fixed sample size for all queries, adaptive sampling [25] (also known as sequential sampling [24]) iteratively samples from data until the accuracy of the estimation satisfies a stopping rule. This approach falls into the runtime sampling category.

Another approach for handling skewed data or highly selective queries is to sample without uniformity. This includes biased sampling [30] or stratified sampling [13], the idea of which is to partition data into non-overlapping clusters of different density, and then assign different weights to the sample tuples from clusters of different density. The density distributions are either estimated through a pilot sampling phase, or from prior knowledge of the queries together with some statistics on the data.

The work by Acharya, Gibbons and Poosala [12] is similar to our approach. Their sampling mechanism is intended for efficient approximate answering of aggregation queries, and their approach is to partition data according to the prior knowledge of grouping attributes, and judiciously assign sample quota among all the group partitions to minimize estimation variance. However, their approach, like those of other biased or stratified sampling techniques for answering group-by queries, requires the groups be non-overlapping. This is because they need to sum up estimates from the groups in the end. One way to apply their approach for our problem is to partition the whole data into non-overlapping regions according to all users'

[4] available from http://www.ipums.org

access controls, and add up estimates from several regions that belongs to a user. However, for U users, we may end up having 2^U non-overlapping regions to compute. Moreover, groups of different sizes have different importance in answering a query, hence Acharya et al.'s work focus on balancing the weights of groups to minimizing the error from summing estimates from groups. On the other hand, our goal is *not* to minimize the error of the sum of all user's estimates, but to minimize the sum of their estimation error bounds.

Some recent work proposes the use of statistics on views [31] as well as *sample views* [26], which consist of a sample of tuples drawn from a particular view. The set of tuples accessible to a given user from a given target relation can be thought of as a user-specific view defined by the user's access control predicate. From this perspective, the samples drawn by the PSALM technique are sample views, and could potentially be exploited by the mechanism described by Larson et al. [26]. However, their work does not consider the problem of determining *which sample views* to define. The sample view definitions assumed to be given, and the emphasis is on how to maintain and exploit the sample views. In contrast, PSALM is about choosing which sample views to define, given some information about static predicates (the access control predicates) that appear repeatedly in many queries.

IX. Conclusion

In this paper, we have proposed PSALM, which is a technique for cardinality estimation in the presence of fine-grained access controls. PSALM exploits knowledge of user access control predicates to identify a set of samples to be used to estimate the cardinalities of queries over access-controlled data. We have shown the PSALM can produce more accurate estimates than those produced by multidimensional histograms and those produced by simpler sampling techniques that do not exploit knowledge of the access control predicates. Although PSALM is motivated by the need for accurate cardinality estimates in the presence of fine-grained access controls, PSALM is potentially applicable in any scenario in which relatively static and repeatedly used query predicates can be identified.

References

[1] S. Rizvi, A. Mendelzon, S. Sudarshan, and P. Roy, "Extending Query Rewriting Techniques for Fine-Grained Access Control," in *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2004, pp. 551–562.

[2] *Data Classification with Oracle Label Security*, Oracle, available at http://www.oracle.com/technology/deploy/security/database-security/pdf/twp_security_db_ols_10gr2.pdf.

[3] *The Virtual Private Database in Oracle9ir2*, Oracle, available at http://otn.oracle.com/deploy/security/oracle9ir2/pdf/vpd9ir2twp.pdf.

[4] *Label-based Access Control (LBAC) Overview*, IBM, available at http://publib.boulder.ibm.com/infocenter/db2luw/v9/topic/com.ibm.db2.udb.admin.doc/doc/c0021114.htm.

[5] *Implementing Row- and Cell-Level Security in Classified Databases Using SQL Server 2005*, Microsoft, 2005, available at http://download.microsoft.com/download/4/7/a/47a548b9-249e-484c-abd7-29f31282b04d/RowCellLvlSecSQL.doc.

[6] I. F. Ilyas, V. Markl, P. Haas, P. Brown, and A. Aboulnaga, "CORDS: Automatic Discovery of Correlations and Soft Functional Dependencies," in *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2004, pp. 647–658.

[7] A. Aboulnaga and S. Chaudhuri, "Self-tuning Histograms: Building Histograms Without Looking At Data," in *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 1999, pp. 181–192.

[8] N. Bruno, S. Chaudhuri, and L. Gravano, "STHoles: A Multidimensional Workload-Aware Histogram," in *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2001, pp. 211–222.

[9] J.-H. Lee, D.-H. Kim, and C.-W. Chung, "Multi-dimensional Selectivity Estimation Using Compressed Histogram Information," in *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 1999, pp. 205–214.

[10] V. Poosala and Y. E. Ioannidis, "Selectivity Estimation Without the Attribute Value Independence Assumption," in *Proc. 23th Int. Conf. on Very Large Data Bases*, 1997, pp. 486–495.

[11] Y. Diao, P. M. Fischer, M. J. Franklin, and R. To, "YFilter: Efficient and Scalable Filtering of XML Documents," in *Proc. 18th Int. Conf. on Data Engineering*, 2002, pp. 341–353.

[12] S. Acharya, P. B. Gibbons, and V. Poosala, "Congressional Samples for Approximate Answering of Group-by Queries," in *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2000, pp. 487–498.

[13] S. Chaudhuri, G. Das, and V. Narasayya, "A Robust, Optimization-Based Approach for Approximate Answering of Aggregate Queries," in *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2001, pp. 295–306.

[14] F. Olken and D. Rotem, "Random Sampling From Database Files: A Survey," in *Proc. of the 5th International Conf. on Statistical and Scientific Database Management*. New York, NY, USA: Springer-Verlag New York, Inc., 1990, pp. 92–111.

[15] R. Gemulla, W. Lehner, and P. J. Haas, "A Dip in the Reservoir: Maintaining Sample Synopses of Evolving Datasets," in *Proc. 32th Int. Conf. on Very Large Data Bases*, 2006, pp. 595–606.

[16] *Bell Numbers*, Wolfram Mathworld, available at http://mathworld.wolfram.com/BellNumber.html.

[17] H. Zhang, N. Zhang, K. Salem, and D. Zhuo, "Compact Access Control Labeling for Efficient Secure XML Query Evaluation," *Journal of Data and Knowledge Engineering*, vol. 60, no. 2, pp. 326–344, 2007.

[18] D. Zhuo, "Fine-Grained Access Control for XML," Master's thesis, University of Waterloo, http://etd.uwaterloo.ca/etd/dhzhuo2003.pdf, 2003.

[19] "Jaccard Similarity Coefficient," available at http://en.wikipedia.org/wiki/Jaccard_index.

[20] P. Crescenzi and V. Kann, *A Compendium of NP Optimization Problems*. Springer Verlag, 1999, p. 13.

[21] B.-U. Pagel, H.-W. Six, H. Toben, and P. Widmayer, "Towards an Analysis of Range Query Performance in Spatial Data Structures," in *Proc. 12th ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems*, 1993, pp. 214–221.

[22] R. Motwani and P. Raghavan, *Randomized algorithms*. New York, NY, USA: Cambridge University Press, 1995.

[23] Y. Matias, J. S. Vitter, and M. Wang, "Wavelet-based Histograms for Selectivity Estimation," in *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 1998, pp. 448–459.

[24] P. J. Haas and A. N. Swami, "Sequential Sampling Procedures for Query Size Estimation," in *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 1992, pp. 341–350.

[25] R. J. Lipton and J. F. Naughton, "Query Size Estimation by Adaptive Sampling," in *Proc. 9th ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems*, 1990, pp. 40–46.

[26] P.-A. Larson, W. Lehner, J. Zhou, and P. Zabback, "Cardinality Estimation Using Sample Views With Quality Assurance," in *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2007, pp. 175–186.

[27] A. El-Helw, I. F. Ilyas, W. Lau, V. Markl, and C. Zuzarte, "Collecting and Maintaining Just-in-Time Statistics," in *Proc. 23st Int. Conf. on Data Engineering*, 2007, pp. 516–525.

[28] A. Aboulnaga, P. J. Haas, S. Lightstone, G. M. Lohman, V. Markl, I. Popivanov, and V. Raman, "Automated Statistics Collection in DB2 Stinger," in *Proc. 30th Int. Conf. on Very Large Data Bases*, 2004, pp. 1146–1157.

[29] B. Epstein, "Introduction to Statistical Analysis," *SIAM Review*, vol. 1, no. 1, pp. 75–77, 1959.

[30] C. Palmer and C. Faloutsos, "Density Biased Sampling: An Improved Method for Data Mining and Clustering," in *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2000, pp. 82–92.

[31] C. A. Galindo-Legaria, M. Joshi, F. Waas, and M.-C. Wu, "Statistics on Views," in *Proc. 29th Int. Conf. on Very Large Data Bases*, 2003, pp. 952–962.