A Robust, Optimization-Based Approach for Approximate Answering of Aggregate Queries

Surajit Chaudhuri
Microsoft Research
One Microsoft Way,
Redmond, WA 98052
+1-(425) 703-1928
surajitc@microsoft.com

Gautam Das
Microsoft Research
One Microsoft Way,
Redmond, WA 98052
+1-(425) 706-8979
gautamd@microsoft.com

Vivek Narasayya
Microsoft Research
One Microsoft Way,
Redmond, WA 98052
+1-(425) 703-2616
viveknar@microsoft.com

ABSTRACT

The ability to approximately answer aggregation queries accurately and efficiently is of great benefit for decision support and data mining tools. In contrast to previous sampling-based studies, we treat the problem as an *optimization* problem whose goal is to minimize the error in answering queries in the given workload. A key novelty of our approach is that we can tailor the choice of samples to be robust even for workloads that are "similar" but not necessarily identical to the given workload. Finally, our techniques recognize the importance of taking into account the variance in the data distribution in a principled manner. We show how our solution can be implemented on a database system, and present results of extensive experiments on Microsoft SQL Server 2000 that demonstrate the superior quality of our method compared to previous work.

1 INTRODUCTION

In recent years, decision support applications such as On Line Analytical Processing (OLAP) and data mining for analyzing large databases have become popular. A common characteristic of these applications is that they execute aggregation queries on large databases, which can often be expensive and resource intensive. Therefore, the ability to obtain approximate answers to such queries accurately and efficiently can greatly benefit the scalability of these applications. One approach to address this problem is to use *precomputed samples* of the data instead of the complete data to answer the queries. While this approach can give approximate answers very efficiently, it is easy to see that identifying an appropriate precomputed sample that avoids large errors on an *arbitrary* query is virtually impossible, particularly when we take into account the fact that queries may involve selections, GROUP BY and join. To minimize the effects of this problem, previous studies have proposed using the *workload* to guide the process of selecting samples [1,6,11]. The hope is that by picking a sample that is tuned to the given workload, we can ensure acceptable error at least for queries in the workload.

Despite recognizing the importance of workload information in picking samples of the data, previous studies suffer from three significant drawbacks. First, although the proposed solutions have intuitive appeal, the lack of a rigorous problem formulation leads to solutions that are difficult to evaluate theoretically. Second, they do not attempt to formally deal with uncertainty in the expected workload, i.e., when incoming queries are "similar" but not identical to queries in the given workload. Third, most previous studies ignore the variance in the data distribution of the aggregated column(s). As the following example shows, ignoring data variance can lead to extremely poor quality of answers for aggregate functions such as SUM:

Example 1. Consider a relation R containing two columns <ProductId, Revenue> and four records {<1, 10>, <2, 10>, <3, 10>, <4, 1000>}. Assume that we are allowed to use a sample S of two records from R to answer the query Q: SELECT SUM(Revenue) FROM R. We answer a query by running it against S and scaling the result by a factor of two (since we are using a 50% sample). Consider a sample S_1 = {<1, 10>, <3, 10>}. The estimated answer for Q using S_1 is 40, which is a severe underestimate of the actual answer (1030). Now consider another sample S_2 = {<1,10>, <4,1000>}. The estimated answer for Q using S_2 is 2020, which is a significant overestimate. Thus, large variance in the aggregate column can lead to large relative errors.

In contrast to most previous sampling-based studies, in this paper, we formulate the problem of precomputing a sample as an *optimization* problem, whose goal is to pick a sample that minimizes the error for the given workload. We show that when the actual workload is identical to the given workload (we refer to such a workload as *fixed*), we can achieve dramatically smaller errors using a *deterministic* solution to the optimization problem. Of course, such a solution is not resilient when the actual workload happens to deviate from the given workload. We therefore introduce a generalized model of the workload ("lifted workload") that makes it possible to tune the choice of the sample so that approximate query processing using the sample is effective not only for workloads that are identical to the given workload, but also for workloads that are "similar" to the given workload (i.e., queries that select regions of the data that overlap significantly with the data accessed by the queries in the given workload) – a more realistic scenario.

We formulate selection of the sample for such a workload as a *stratified sampling* problem with the goal to minimize error in estimation of aggregates. Our formulation makes the problem amenable to exploiting known techniques in stratified sampling and optimization. As a consequence, we have developed a robust approach to the problem of approximate query processing of SPJ queries with GROUP BY and aggregation. We have implemented our solutions on Microsoft SQL Server 2000, addressing the pragmatic issues that are central to an effective solution that can be deployed in a commercial DBMS. The benefits of our systematic approach are amply demonstrated not only by

ACM SIGMOD 2001, May 21-24, 2001, Santa Barbara, California, USA.

theoretical results, but also experimentally on synthetic as well as on a deployed enterprise data-warehouse in our organization.

Some details of our work are omitted in this paper due to lack of space. A complete version of the paper is available in [7]. We begin by discussing related work in Section 2. We present an overview of our architecture for approximate query processing in Section 3. Our deterministic solution for the special case of a *fixed* workload is presented in Section 4. We describe a model for lifting a given workload in Section 5, and formulate the problem of approximate query answering using stratified sampling in Section 6. We present our solution to the optimization problem for single-table selection queries with aggregation in Section 7, and describe extensions necessary for a broader class of queries in Section 8. We describe our implementation and experimental results in Section 9, and conclude in Section 10.

2 RELATED WORK

Both [6] and [11] present a seemingly intuitive idea based on *weighted sampling*. Each record in the relation R to be sampled is tagged with a *frequency* – the number of queries in the workload such that the record must be selected to answer the query. Once tagging is done, an expected number of k records are selected in the sample, where the probability of selecting a record t with frequency f_t is $k*(f_t/\Sigma_u f_u)$. Thus, records that are accessed more frequently have a greater chance of being included inside the sample. However, as the following example shows, this approach can sometimes lead to poor quality. Consider a set of k queries $\{Q_1, \ldots Q_k\}$ (where k is also the size of the sample) that reference disjoint partitions of records in R. Let a few queries reference large partitions and most queries reference very small partitions. Then, by the weighted sampling scheme described above, since most records in the sample will come from the large partitions, with high probability, there will be no records selected from many of the small partitions. Thus, the relative error in answering most of the queries will be large. For this example, as we show in Sections 4 and 7, we can achieve significantly lower error. A novelty of [11] is that it tackles the issue of maintaining and continuously refreshing a sample of records of R after a new query has been processed. However, the paper does not address the issue of variance of data in the aggregate column. Finally, as mentioned in the introduction, a shortcoming common to most previous work is that they do not attempt to formally deal with uncertainty in the expected workload.

The paper [6] attempted to address the problem of variance of data in the aggregate column (see Example 1). The basic idea is that *outliers* of the data (i.e., the records that contribute to high variance in the aggregate column) are collected into a separate index, while the remaining data is sampled using a weighted sampling technique. Queries are answered by running them against both the outlier index as well as the weighted sample, and an estimated answer is composed out of both results. This method too is easily seen to result in worse quality than our approach, since the concept of an outlier index + a (weighted) sample can be viewed as crude approximation of our approach using stratified sampling, where the outliers form their own stratum that is sampled in its entirety. The idea of separately handling outliers has also appeared in the context of applying exploratory data analysis methods on data cubes [3,4].

The *congressional sampling* paper [1] does not deal with data variance. However, they adopt the most principled approach among previous work. They advocate a stratified sampling strategy called *Congress* that tries to simultaneously satisfy a set of GROUP BY queries. Some key concepts of our paper (e.g., the concept of *fundamental regions* that we discuss in Section 4) have been influenced by it. However, their approach is still ad-hoc in the sense that their scheme does not attempt to minimize the error for any of the well-known error metrics.

There has been a large body of work on approximately answering a query by sampling on the fly rather than exploiting a precomputed sample. However, in general, on-the-fly sampling can be expensive, particularly in the presence of join and GROUP BY without extensive availability of statistics (histograms) and/or enhancements to the database engine. Notable examples of on-the-fly sampling techniques are [8,13]. In [13], they also identify enhancements to the database engine needed to support progressive refinement of the approximate answer.

In addition to sampling based methods, there have been other data reduction based approaches to approximate query processing, such as histograms [15,17] and wavelets [5,22,23]. As noted in [22], a general problem with histogram-based approaches is that they incur high storage overhead and construction cost as the dimensionality of the data increases. In [22,23], the authors argued the effectiveness of wavelets for handling aggregations over (high-dimensional) OLAP cubes. More recently, [5] showed how SQL operators can be applied directly on wavelet coefficients to efficiently produce approximate answers. More extensive theoretical and experimental comparisons of data reduction based approaches and sampling based approaches are necessary to identify their relative strengths and weaknesses.

3 ARCHITECTURE FOR APPROXIMATE QUERY PROCESSING

3.1 Preliminaries

We present an overview of our architecture for approximate query processing on a relational database. We consider queries with selections, foreign-key joins and GROUP BY containing aggregation functions such as COUNT, SUM, and AVG. We assume that a pre-designated amount of storage space is available for selecting samples from the database. These samples, possibly in conjunction with other base relations, will be used for answering the queries approximately but efficiently. The techniques for selecting samples can be *randomized* (e.g., we may sample uniformly at random) or *deterministic* (e.g., we may select the best sample that minimizes the total error in the approximate answers).

As with previous sampling-based studies [1,6,11], we have taken the approach of exploiting the available workload (provided as an input) to find samples that work well for queries in the given workload. A workload **W** is specified as a set of pairs of queries and their corresponding weights: i.e., $\mathbf{W} = \{<Q_1, w_1>, \ldots <Q_q, w_q>\}$, where weight w_i indicates the importance of query Q_i in the workload. Without loss of generality, we can assume that the weights are normalized, i.e., $\Sigma_i w_i=1$. In practice, such a workload may be obtained using profiling tools available on most modern DBMSs that for logging queries that execute on the server.

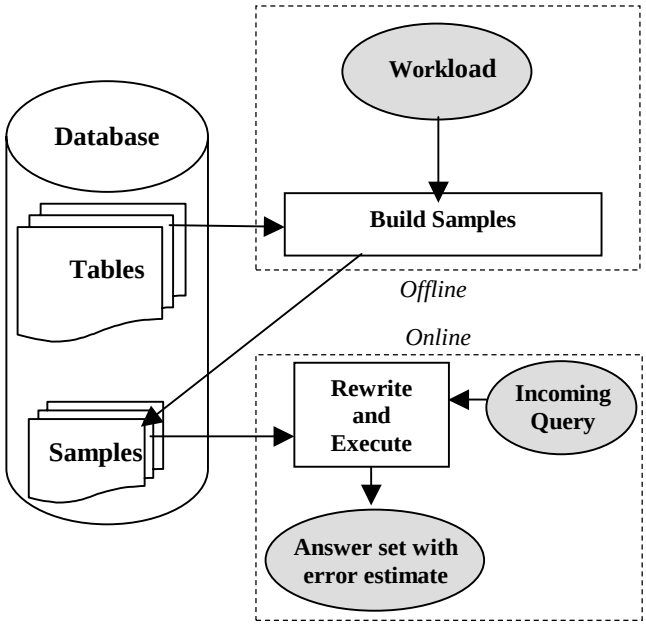

Figure 1. Architecture for Approximate Query Processing.

3.2 Our Architecture

Our architecture for approximate query processing is summarized in Figure 1. The inputs are a database and a workload **W**. For simplicity, we present our architecture for the case of a single relation R. There are two components in our architecture: (1) an *offline* component for selecting a sample of records from relation R, and (2) an online component that (a) rewrites an incoming query to use the sample (if appropriate) to answer the query approximately and (b) reports the answer with an estimate of the error in the answer. The novelty of this paper is in the first component. We present a method for automatically *lifting* a given workload, i.e., quantifying a generalized model of the workload. Our motivation stems from the fact that it is unrealistic to assume that incoming queries in the future will be *identical* to the given workload **W**. The key to lifting the workload is the ability to compute a *probability distribution* $p_{\mathbf{W}}$. of incoming queries, i.e., for any incoming query Q, $p_{\mathbf{W}}(Q)$ is the probability of Q. The subscript indicates that the distribution depends on **W**. Our algorithm then selects a sample that is resilient enough for such a lifted workload. We also show how we can select a sample that minimizes the error of answering queries in the (lifted) workload. This step is labeled "Build Samples" in the figure.

An incoming query is rewritten to run against the samples instead of the base relation. For a multi-relation query, in addition to the samples, we may also reference other base relations to answer the query. As in previous work [1,6,11], we assume that each record in the sample also contains an additional column known as the *ScaleFactor* with each record[1]. The value of the aggregate column of each record in the sample is first scaled up by multiplying with the *ScaleFactor*, and then aggregated. We note that alternative schemes, as in [1], are possible where the *ScaleFactor* column is maintained in a separate relation than the sample. Such schemes incur reduced update and storage overhead at the expense of increased run time overhead. The techniques described in this paper are applicable independent of the specific scheme used. In addition to the approximate answer, as we show in Section 6, we can also report the variance (or even a confidence interval) for the approximate answer.

[1] In general, we allow a small constant number of additional columns with each record (e.g., see deterministic solution in Section 4).

3.3 Error Metrics

We define the error metrics used to determine the quality of an approximate answer to an aggregation query. Suppose the correct answer for a query is y while the approximate answer is y'. We focus on *relative error* (defined as $|y - y'|/|y|$), since that is usually a fairer measure across queries. The squared error of a query (SE(Q)) is $((y - y')/ y)^2$. Now consider a GROUP BY query that induces g groups in the data. Suppose the correct answer for the i*th* group is y_i while the approximate answer is y_i'. The squared error for the query is $(1/g)\ \Sigma_i((y_i - y_i')/ y_i)^2$ (this error measure for a GROUP BY query has also been considered by [1,6]). In other words, a GROUP BY query can be treated as g SELECT queries, each of weight 1/g. Given a probability distribution of queries $p_{\mathbf{W}}$, the mean squared error for the distribution ($MSE(p_W)$) is defined as $\sum_Q p_W(Q)*SE(Q)$, where $p_W(Q)$ is the probability of query Q. The root mean squared error (RMSE), also known as the L_2 error, is defined as the square root of MSE. Other error metrics are possible e.g., using the L_1 metric (defined as the mean error over all queries in the workload) or L_∞ metric (defined as the maximum error over all queries). In this paper, although we optimize for the MSE due to its long tradition in statistics, we can easily extend our techniques to optimize for the L_1 metric. In fact, while our algorithms minimize MSE, we found that these solutions also do very well for the L_1 metric. Since most previous work in this area report the L_1 metric, our experimental comparisons also report the L_1 metric.

4 THE SPECIAL CASE OF A FIXED WORKLOAD

In this section, we present a problem formulation and solution for the special case of a fixed workload, i.e., when the incoming queries are *identical* to the given workload. The motivation for presenting this case is to underscore the benefit of our approach of treating approximate query answering as an *optimization* problem. In fact, as shown below, this problem formulation allows us to use an effective *deterministic* scheme rather than the conventional randomization schemes considered in previous work. For simplicity, we describe the problem for the case of single table selection queries containing the COUNT or SUM aggregate.

4.1 Problem Formulation

Problem: **FIXEDSAMP**
Input: R, **W,** k
Output: A sample of k records (with appropriate additional columns) such that MSE(**W**) is minimized.

We now frame the optimization problem FIXEDSAMP for the case of a fixed workload **W**. Recall that $MSE(p_{\mathbf{W}})$ (Section 3.3) is the mean squared error for the probability distribution of queries $p_{\mathbf{W}}$. MSE(W) is equivalent to $MSE(p_{\mathbf{W}})$ where a query Q has a probability of occurrence of 1 if $Q \in \mathbf{W}$ and 0 otherwise. As described in Section 3.2, we need to associate additional column(s) with each record in the sample to allow scaling the values obtained by running the query on the sample. Observe that

the problem formulation is general in the sense that it allows both randomized as well as deterministic solutions.

Before presenting our solution to FIXEDSAMP, we first define the key concept of *fundamental regions* of a relation induced by a workload. Fundamental regions are important because they play a crucial role in determining an appropriate sample for the given workload. In fact, the concept of fundamental regions is also important in the context of our randomized sampling scheme that appears in Section 7.

4.2 Fundamental Regions

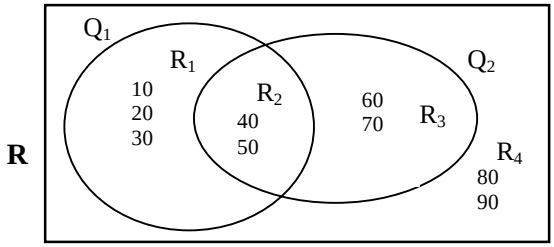

Figure 2. Fundamental Regions

For a given relation R and workload **W**, consider partitioning the records in R into a minimum number of regions R_1, R_2, …, R_r such that for any region R_j, each query in **W** selects either all records in R_j or none. These regions are the *fundamental regions* of R induced by **W**. For example, consider a relation R (with aggregate column C) containing nine records (with C values 10, 20, …, 90), as shown in Figure 2. Let **W** consist of two queries, Q_1 (which selects records with C values between 10 and 50) and Q_2 (which selects records with C values between 40 and 70). These two queries induce a partition of R into four fundamental regions, labeled R_1, … R_4.

The concept of *finest partitioning* into groups in [1] is similar to the concept of fundamental regions. In general the total number of fundamental regions *r* depends on R and **W** and is upper-bounded by $\min(2^{|W|}, n)$ where *n* is the number of records in R. The algorithmic and implementation details of how to identify fundamental regions efficiently are discussed in Section 7.3.1.

4.3 Solution for FIXEDSAMP

We present a deterministic algorithm called FIXED for solving FIXEDSAMP. Briefly, the algorithm has three steps. The first step identifies all fundamental regions. The second step selects the sample by picking exactly one record from each "important" fundamental region. The third step assigns appropriate values to additional columns in the sample records. We elaborate on these steps below.

Step1 (*Identify Fundamental Regions*): The first step is to identify the fundamental regions in R induced by the given workload **W**. Let *r* be the number of fundamental regions.

After Step 1, two cases arise that need to be separately processed: Case A ($r \leq k$) and Case B ($r > k$).

Case A ($r \leq k$): For this case our algorithm selects a sample that can answer queries *without any errors*. Details are as follows.

Step 2A (*Pick Sample Records*): We select the sample by picking exactly one record from each fundamental region. Thus for the example in Figure 2, we may pick the records with C values 10, 40, 60, and 80, i.e. one record from each fundamental region.

Step 3A (*Assign Values to Additional Columns*): The idea is that each sample record can be used to "summarize" all records from the corresponding fundamental region, without incurring any error. More precisely, for a workload consisting of only COUNT queries, we need a single additional column in the sample records (called *RegionCount*), in which we store the count of the number of records in that fundamental region. This allows us to answer a COUNT query *without any errors* by running it against the sample and simply summing up the *RegionCount* column of records selected from the sample by the query. For example, if the queries in Figure 2 were COUNT queries, the sample records chosen in Step 2A will contain an extra *RegionCount* column with values 3, 2, 2, and 2 respectively. Likewise, for a workload consisting only of SUM queries, we need a single additional column in the sample (called *AggSum*) that contains the sum of the values in the aggregate column for records in that fundamental region. For example, if the queries in Figure 2 were SUM queries, the sample records chosen in Step 2A will contain an extra *AggSum* column with values 60, 90, 130, and 170 respectively. If the workload contains a mix of COUNT and SUM queries, we need both the *RegionCount* and the *AggSum* columns. Note that if we include both these columns, we can also answer AVG queries.

Case B ($r > k$): This is a more difficult case. Our algorithm selects a sample that tries to minimize the errors in queries.

Step 2B (*Pick Sample Records*): Since $r > k$, we select *k* regions and then pick one record from each of the selected regions. Our heuristic for selecting *k* regions is to sort all *r* regions by their *importance* and then select the top *k*. The importance of region R_j is defined as $f_j * n_j^2$, where f_j is the sum of the weights of all queries in **W** that select the region, and n_j is the number of records in the region. The intuition is that f_j measures the weights of the queries that are affected by R_j while n_j^2 measures the effect on the (squared) error by not including R_j. While more complicated measures of importance are possible, in our experiments we found that the above heuristic does very well. We then pick exactly one record from each selected fundamental region.

Step 3B (*Assign Values to Additional Columns*): Next, for the selected sample records, we determine the values of the *RegionCount* and *AggSum* columns. We could of course naively do exactly what was done in Step 3A, i.e. we store in the *RegionCount* and *AggSum* columns of each sample record the count and sum of the records of the corresponding fundamental region. However, note that the extra column values of a sample record are *not required* to have any obvious relationship with some characteristic of the corresponding fundamental region; all we care is that they contain appropriate values so that the error for the workload is minimized.

Thus, we view the problem of assigning values to the *RegionCount* and *AggSum* columns of the *k* records selected in Step 2B as the following optimization problem. We have $2*k$ unknowns: $\{RC_1, \ldots, RC_k\}$ and $\{AS_1, \ldots AS_k\}$. MSE(W) can be expressed as a quadratic function of these $2*k$ unknowns. We minimize this function by partially differentiating with each variable and setting each result to zero. This gives rise to $2*k$ simultaneous (sparse) linear equations, which we solve using an iterative technique (based on the Gauss-Seidel method [12]). In our experiments, (Section 9), we see that FIXED is significantly more accurate than all randomized schemes for the given workload. We note that the disadvantage of this deterministic

method is that a per-query (probabilistic) error guarantee is not possible.

Observe that if the incoming query is not identical to a query in the given workload (a realistic scenario), using FIXED can result in unpredictable errors. Therefore, our goal is to incorporate a measure of robustness in our solution by optimizing for a more generalized model of the workload that would allow incoming queries to be similar but not necessarily identical to the given workload.

5 LIFTING WORKLOAD TO QUERY DISTRIBUTIONS

As mentioned earlier, we would like our approximate query processing scheme to not only perform well for incoming queries that exactly match one of the queries in the given workload, but also be resilient to the situation when an incoming query is "similar" but not identical to queries in the workload. In this section we tackle one aspect of the problem, i.e., defining this notion of similarity. More formally, we show how given **W**, we can define a *lifted workload* $p_{\mathbf{W}}$, i.e., a probability distribution of incoming queries. Intuitively, for any query Q' (not necessarily in **W**), $p_{\mathbf{W}}(Q')$ should be related to the amount of similarity (dissimilarity) of Q' to the workload: high if Q' is similar to queries in the workload, and low otherwise. In Sections 7 and 8 we show how to leverage such a probability distribution in our approximate query processing solution.

Our notion of similarity between queries is not concerned with syntactic similarity of query expressions. Rather, we say that two queries Q' and Q are similar if the records selected by Q' and Q have significant overlap. We focus on the case of single-table selection queries with aggregation containing either the SUM or COUNT aggregate (this intuition is refined for GROUP BY and join queries in Section 8). Let us consider the simplest case when the workload **W** consists of exactly one query Q on relation R. Let R_Q be the records selected by Q. Our objective is to define the distribution $p_{\{Q\}}$ (i.e., for $p_{\mathbf{W}}$, where **W** = {<Q,1.0>}). Since for the purposes of lifting, we are only concerned with the set of records selected by a query and not the query itself, we make a change in notation for convenience: instead of mapping queries to probabilities, $p_{\{Q\}}$ maps subsets of R to probabilities[2]. For all $R' \subseteq R$, $p_{\{Q\}}(R')$ denotes the probability of occurrence of any query that selects exactly the set of records R'.

For the moment, assume two parameters δ ($\frac{1}{2} \leq \delta \leq 1$) and γ ($0 \leq \gamma \leq \frac{1}{2}$) have been specified. Informally, these parameters define the degree to which the workload "influences" the query distribution. More formally, for any given record inside (resp. outside) R_Q, the parameter δ (resp. γ) represents the probability that an incoming query will select this record.

Given these two parameters, we can now derive $p_{\{Q\}}(R')$ for any $R' \subseteq R$ (i.e. the probability of occurrence of any query that exactly selects R'). Figure 3 shows a Venn diagram of R, R_Q and R', where n_1, n_2, n_3, and n_4 are the counts of records in the regions indicated. Equation 1 shows the derivation of $p_{\{Q\}}(R')$. Note that when n_2 or n_4 are large (i.e., the overlap is large), $p_{\{Q\}}(R')$ is high (i.e. queries that select R_Q are likely to occur), whereas when n_1 or n_3 are large (i.e. the overlap is small), $p_{\{Q\}}(R')$ is low (i.e. queries that select R_Q are unlikely to occur). Once $p_{\{Q\}}$ has been defined, $p_{\mathbf{W}}$ can be easily derived, as shown in Equation 2.

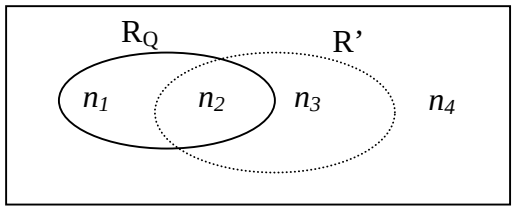

Figure 3

$$p_{\{Q\}}(R') = \delta^{n_2}(1-\delta)^{n_1}\gamma^{n_3}(1-\gamma)^{n_4} \qquad (1)$$

$$p_{\mathrm{W}}(R') = \sum_{i}^{q} w_i \, p_{\{Q_i\}}(R') \qquad (2)$$

Let us now discuss the problem of setting the parameters δ and γ. As mentioned earlier, the parameters define the degree to which the workload **W** influences the query distribution $p_{\mathbf{W}}$. We elaborate on this issue by analyzing the effects of (four) different boundary settings of these parameters.

1. $\delta \to 1$ and $\gamma \to 0$: implies that incoming queries are identical to workload queries.
2. $\delta \to 1$ and $\gamma \to \frac{1}{2}$: implies that incoming queries are supersets of workload queries.
3. $\delta \to \frac{1}{2}$ and $\gamma \to 0$: implies that incoming queries are subsets of workload queries.
4. $\delta \to \frac{1}{2}$ and $\gamma \to \frac{1}{2}$: implies that incoming queries are unrestricted.

Using the above scenarios as guidelines, it may be possible for skilled database administrators to analyze their workload patterns, and manually set the parameters to values that best model their workloads. However, we also present a simple automated approach for parameter setting. The basic idea is to split the available workload into two sets, the *training* workload and the *test* workload. The parameters are selected using a two-dimensional grid search approach (based on [19]) such that the lifted training workload (under these settings) most closely fits the test workload. The implementation details of this method appear in Section 9.1. The grid search approach is effective and scalable with data size for low dimensional optimization problems such as ours, and our experiments (Section 9) indicate the approach is promising. We are also investigating alternative approaches such as randomized search and gradient descent.

The above represents a simple first attempt at lifting workloads in a rigorous manner. It is similar to kernel density estimation techniques for estimating probability distributions from samples [16,18]. The problem of automatically setting parameters δ and γ is similar to the problem of bandwidth selection in kernel density estimation. We are investigating whether known techniques for bandwidth estimation (bootstrap, cross-validation [18]) can be adapted in a scalable manner. Other methods for lifting a workload need to be studied in the future, e.g., modeling the query distribution as a mixture of Gaussians. In fact, the problem of lifting a workload is really orthogonal to the problem of

[2] This notation makes it convenient to give a single probability to the (infinite) set of queries that only syntactically differ in their WHERE clauses, yet select the same R'. Note that the domain of $p_{\{Q\}}$ is finite, i.e. the power set of **R**.

approximate query processing, and we expect it to find applications in other areas.

In the next few sections, we develop an approximate query processing scheme, which will attempt to minimize the MSE of the lifted workload, i.e., for $p_{\mathbf{W}}$ (which depends on **W**, δ and γ).

6 RATIONALE FOR STRATIFIED SAMPLING

We now state the problem of identifying an appropriate sample as a formal optimization problem. For simplicity, we state the problem when the workload contains queries that reference a single relation R. The formulation can be easily extended for multi-table queries (see Section 8).

> ***Problem*: SAMP**
> ***Input***: R, $p_{\mathbf{W}}$ (a probability distribution function specified by **W**), and *k*
> ***Output***: A sample of *k* records, (with the appropriate additional column(s)) such that the MSE($p_{\mathbf{W}}$) is minimized.

In the above formulation, $p_{\mathbf{W}}$ is any probability distribution function derived from the given workload **W**. For example, the lifting model presented in Section 5 can be used to obtain $p_{\mathbf{W}}$. In this section we show why uniform sampling cannot be effectively applied to SAMP, and justify our approach of adapting *stratified sampling* [10] to solve this problem. Stratified sampling is a well-known generalization of uniform sampling where a population is partitioned into multiple strata and samples are selected uniformly from each stratum, with "important" strata contributing relatively more samples.

We first observe that the error incurred by uniformly sampling a population of numbers is proportional to the variance of the population and inversely proportional to the sample size [10]. More precisely, consider a population, i.e. a set of numbers R = $\{y_1,\ldots, y_n\}$. Let the average be *y*, the sum be Y and the variance be S^2. Suppose we uniformly sample *k* numbers. Let the mean of the sample be μ. The quantity μ is an unbiased estimator for *y*, i.e., $E[\mu] = y$; the variance (i.e., squared error) in estimating *y* is $E[(\mu - y)^2] = S^2/k$.

Now consider the following selection query with aggregation on relation R defined in Example 1 (Section 1). Q_1 = SELECT COUNT(*) FROM R WHERE ProductId IN (3,4). Recall that R is the relation {<1, 10>, <2,10>, <3, 10>, <4,1000>}. We define the *population* of a query Q (denoted by POP_Q) on a relation R as a set of size |R| that contains the value of the aggregated column that is selected by Q, or 0 if the record is not selected. By this definition, POP_{Q1} = {0, 0, 1, 1}. Observe that POP_{Q1} has a mix of 1's and 0's and thus has a non-zero variance. Thus, a uniform sampling of POP_{Q1} would be a poor choice for this problem since it would incur non-zero error. However, if we partition R into two *strata* {<1, 10>, <2,10>} and {<3, 10>, <4,1000>}, we effectively partition POP_{Q1} into two strata {0, 0} and {1, 1}. Each stratum now has zero variance, and a stratified sampling strategy that selects at least one sample from each stratum will estimate Q_1 with zero error.

Note however, that this particular stratification may not work well for a different COUNT query whose population has a different distribution of 1s and 0s. For example, consider a query Q_2 = SELECT COUNT(*) FROM R WHERE ProductId IN (1,2,3). POP_{Q2} = {1, 1, 1, 0} and is different from POP_{Q1}. As can be seen by this example, each query defines its own population of the same relation R, and therefore the challenge is to adapt stratified sampling so that it works well for all queries. An effective scheme will need to stratify the relation such that the expected variance over all queries in each stratum is small, and allocate more samples to strata with larger expected variances.

For SUM queries, stratification is also governed by the additional problem of variance in the aggregate column. For example, consider query Q_3 = SELECT SUM(Revenue) FROM R WHERE ProductID IN (1,4). POP_{Q3} = {10, 0, 0, 1000} and therefore has large variance.

Thus, a stratified sampling scheme partitions R into *r* strata containing $n_1, ., n_r$ records (where $\Sigma n_j = n$), with $k_1, \ldots, k_r$ records uniformly sampled from each stratum (where $\Sigma k_j = k$). As mentioned in Section 3.2, the scheme also associates a *ScaleFactor* with each record in the sample. Queries are answered by executing them on the sample instead of R. For a COUNT query, the *ScaleFactor* entries of the selected records are summed, while for a SUM(*y*) query the expression *y*ScaleFactor* is summed. If we also wish to return an error guarantee with each query, then instead of *ScaleFactor*, we have to keep track of each n_j and k_j individually for each stratum.

7 SOLUTION FOR SINGLE-TABLE SELECTION QUERIES WITH AGGREGATION

We now present STRAT, our solution to the problem SAMP (Section 6) for workloads consisting of single-table selection queries with aggregation. In Section 8, we show how to extend STRAT for aggregation queries with join, nested sub-queries, and GROUP BY. Our solution consists of three steps. The first step, which we refer to as *stratification*, is determining (a) how many strata *r* to partition relation R into, and (b) the records from R that belong to each stratum. At the end of this step we have *r* strata $R_1, \ldots R_r$ containing $n_1, \ldots n_r$ records such that $\Sigma n_j = n$. The second step, called *allocation*, determines how to divide *k* (the number of records available for the sample) into integers $k_1, \ldots, k_r$ across the *r* strata such that $\Sigma k_j = k$. The third step, referred to as the *sampling* step, uniformly samples k_j records from stratum R_j to form the final sample of *k* records. The sample so created is then used at runtime to approximately answer queries. The heart of the algorithm is in the first two steps, which are designed to minimize the errors in approximately answering queries in the lifted workload ($p_{\mathbf{W}}$). The third step is straightforward, and can be accomplished with one scan of relation R. We present STRAT for queries containing the COUNT aggregate, and then describe the extensions necessary to deal with the more challenging SUM aggregate. We then discuss the key implementation issues.

7.1 Solution for COUNT Aggregate

7.1.1 Stratification

It may appear that the problem of stratification of R for a given workload **W** of COUNT queries is intractable since when *r* is not known, there are an exponential number of ways of stratifying R. However, the following lemma tells us that it is enough to partition R into fundamental regions (Section 4.2) and treat each region as a stratum. For details of the proof see [7].

Lemma 1: For a workload **W** consisting of COUNT queries, the fundamental regions represent an optimal stratification.

7.1.2 Allocation

The key remaining challenge is how to allocate the k records across the r fundamental regions (strata). Our main idea is to treat this problem as an optimization problem whose goal is to minimize the error over queries in $p_{\mathbf{W}}$. Observe that this is a *significant* point of departure compared to most previous work in this area, where this allocation step is done in an intuitive but informal manner. We assume that $k_1,\ldots, k_r$, are unknown variables such that $\Sigma k_j = k$. We leverage the following two results to express $MSE(p_{\mathbf{W}})$ as a function of these variables and then select values for these variables that minimizes $MSE(p_{\mathbf{W}})$. First, using Equation 2, it is easy to see that the $MSE(p_{\mathbf{W}})$ can be expressed as a weighted sum of the MSE of each query in the workload (as stated by the following lemma):

Lemma 2: $MSE(p_{\mathbf{W}}) = \Sigma_i\, w_i\, MSE(p_{\{Q\}})$

Next, for any $Q \in W$, we express $MSE(p_{\{Q\}})$ as a function of the k_j's. Although obtaining a concise yet exact expression for this function is more difficult, under *large population* assumptions (i.e., when n, the number of records in R, is large), the following lemma (one of the principal results of this paper) shows how to obtain a succinct approximation for $MSE(p_{\{Q\}})$. In our experiments, we have found that this formula for $MSE(p_{\{Q\}})$ has yielded excellent approximation even when n is relatively small.

Lemma 3: For a COUNT query Q in **W,** let

$$ApproxMSE(p_{\{Q\}}) = \frac{\sum_{R_j \subseteq R_Q} \frac{n_j^2}{k_j}\delta(1-\delta) + \sum_{Rj \subseteq R \setminus R_Q} \frac{n_j^2}{k_j}\gamma(1-\gamma)}{\left(\sum_{R_j \subseteq R_Q} \delta n_j + \sum_{R_j \subseteq R \setminus R_Q} \gamma m_j\right)^2}$$

Then

$$\lim_{n\to\infty} \frac{ApproxMSE(p_{\{Q\}})}{MSE(p_{\{Q\}})} = 1$$

Outline of Proof: We provide an outline of the proof for the case where we assume *each* n_j is large (the proof for the more general case where we assume only n to be large appears in [7]). Let Q' be a query randomly drawn from the distribution $p_{\{Q\}}$. The number of records selected by Q' in each fundamental region follows a binomial distribution. Since each n_j is large, an overwhelming number of queries from the distribution $p_{\{Q\}}$ will select approximately $\delta * n_j$ (resp. $\gamma * n_j$) records from R_j, where R_j is a fundamental region inside (resp. outside) R_Q. Thus, $MSE(p_{\{Q\}})$ can be approximated as the MSE of all such queries since the contribution from the other queries is negligible. Consider the j*th* term in the left summation in the numerator. It represents the expected squared error in estimating the count of $(R_{Q'} \cap R_j)$, i.e., in estimating the sum of the portion of $POP_{Q'}$ that corresponds to R_j (see Section 6). This may be derived using the error formula presented in Section 6. Similarly, the right summation in the numerator represents the expected squared error in estimating the count of $(R_{Q'} \cap (\mathbf{R} \setminus R_Q))$. Thus the numerator represents the expected squared error in estimating the count of $R_{Q'}$. Dividing by the denominator represents the expected *relative* squared error in estimating the count of $R_{Q'}$. ■

Now that we have an (approximate) formula for $MSE(p_{\{Q\}})$, we can express $MSE(p_{\mathbf{W}})$ as a function of the variables $k_1,\ldots,k_r$, using the result from the following corollary, which is obtained by combining Lemmas 2 and 3.

Corollary 1: $MSE(p_{\mathbf{W}}) = \Sigma_j(\alpha_j / k_j)$, where each α_j is a function of $n_1,\ldots,n_r$, δ, and γ.

Intuitively, α_j captures the "importance" of a region; it is positively correlated with n_j as well as the frequency of queries in the workload that access R_j. Now that we have expressed $MSE(p_{\mathbf{W}})$ as a function of the unknown k_j's, we are ready to minimize it.

Lemma 4: $\Sigma_j(\alpha_j / k_j)$ is minimized subject to $\Sigma_j k_j = k$ if $k_j = k * (sqrt(\alpha_j) / \Sigma_i sqrt(\alpha_i))$.

Proof: We first eliminate one of the variables, say k_r, by replacing it with $k - (k_1 + \ldots + k_{r-1})$. If we partially differentiate $\Sigma_j(\alpha_j / k_j)$ by $k_1,\ldots, k_{r-1}$ respectively and set each derivative to zero, this results in $r-1$ equations. These equations can be easily solved to prove the lemma. ■

Lemma 4 provides us with a closed-form and computationally inexpensive solution to the allocation problem since α_j depends only on δ, γ and the number of tuples in each fundamental region. The proof exploits a technique similar to other well-known methods for minimizing functions of the form $\Sigma_j(\alpha_j / k_j)$ that arise in different contexts (e.g. [2,10]). Note that an admissible solution in our case requires that each k_j is an integer > 0. We discuss this issue in Section 7.3.3. For now, we assume that STRAT completes its allocation by dividing k into $k_1,\ldots,k_r$ according to Lemma 4.

7.2 Solution for SUM Aggregate

We now highlight the extensions to the above solution required for queries containing only the SUM aggregate. The key difference arises due to the fact that for SUM, we also need to take into account the variance of the data in the aggregated column (see Example 1 in Section 1). The first effort to deal with variance in data for approximate query processing was the outlier-indexing technique presented in [6]. We use a more general and principled approach that adapts techniques from statistics for dealing with large variance. We note that both the stratification and allocation steps for the SUM are sufficiently different from COUNT, and need to be revisited.

7.2.1 Stratification

If we use the same stratification as in the COUNT case, i.e., strata = fundamental regions, we may get poor solutions for SUM since each stratum now may have large internal variance in the values of the aggregate column. Therefore, we use a bucketing technique where we further divide fundamental regions with large variance into a set of finer regions, each of which has significantly lower internal variance. We then treat these finer regions as the strata. Within a new stratum the aggregate column values of records are close to one another. We borrow from statistics literature an approximation of the optimal *Neymann Allocation* technique for minimizing variance [10], and use it to divide each fundamental region further into h finer regions, thus generating a total of $h*r$

strata. This can be achieved in a single scan of the R. Although in principle, the larger the *h* the lower the internal variance of the new regions, we set the value *h* to 6 as suggested in [10].

7.2.2 Allocation

The structure of the allocation step is similar to COUNT, i.e., it is expressed as an optimization problem with $h*r$ unknowns $k_1, \ldots, k_{h*r}$. However, there is a key difference. For SUM (unlike COUNT), the specific values of the aggregate column, as well as the variance of values in each region influence MSE($p_{\{Q\}}$). Let y_j (Y_j) be the average (sum) of the aggregate column values of all records in region R_j. Since the variance within each region is small (due to stratification), we can assume that each value within the region can be approximated as y_j. Now observe that a query Q' randomly drawn from the distribution $p_{\{Q\}}$ picks up an expected $\delta*n_j$ (resp. $\gamma*n_j$) records from each finer region R_j that is inside (outside) Q. Therefore, the quantity $y_j^2*\delta(1-\delta)$ (resp. $y_j^2*\gamma(1-\gamma)$) is a good approximation for the expected variance of the portion of $POP_{Q'}$ (see Section 6) that corresponds to a region R_j inside (resp. outside) Q. We now present an approximate formula for MSE($p_{\{Q\}}$) for a SUM query Q in **W**:

$$\frac{\sum_{R_j \subseteq R_Q} \frac{n_j^2}{k_j} y_j^2 \delta(1-\delta) + \sum_{R_j \subseteq R \setminus R_Q} \frac{n_j^2}{k_j} y_j^2 \gamma(1-\gamma)}{\left(\sum_{R_j \subseteq R_Q} \delta Y_j + \sum_{R_j \subseteq R \setminus R_Q} \gamma Y_j \right)^2}$$

The above formula is effective for MSE($p_{\{Q\}}$), except in the following circumstance. Consider a relation R that has a mix of positive and negative numbers, and furthermore suppose a subset R' exists whose SUM is close to zero (i.e. the negative values cancel the positive values), but whose variance is large. Even though a query Q' that selects R' may have a small probability of occurrence in the lifted distribution, if not answered exactly, its relative error can become infinite. In fact, most sampling methods cannot handle such queries, and these queries need to be recognized and processed separately. This situation does not arise if, for example, the values in R are all strictly positive (or strictly negative), or R contains positive as well as negative numbers but does not contain subsets such as R'. We note that in our experiments the above formula has consistently worked well.

As with COUNT, MSE($p_{\mathbf{W}}$) for SUM is functionally of the form $\Sigma_j(\alpha_j / k_j)$, and α_j depends on the same parameters $n_1, \ldots n_{h*r}$, δ, and γ (see Corollary 1) (although the exact value of α_j is different from COUNT). We can therefore use the same procedure for minimization as in Lemma 4.

7.3 Pragmatic Issues

7.3.1 Identifying Fundamental Regions

During the offline process of building a sample, we use a technique that we refer to as *tagging* to identify fundamental regions in relation R for a workload **W** consisting of selection queries with aggregation. Tagging (logically) associates with each record $t \in R$ an additional column called *TagColumn* (of type varchar) that contains the list of queries in **W** that reference *t*. In our implementation, rather than adding *TagColumn* to R, we separate this column out into a different relation R' for two reasons. First, from a pragmatic standpoint, users do not want to change the schema of their tables if avoidable. Second, we found that it is significantly faster (3X-5X in our experiments) to update the *TagColumn* in a separate relation R'. Records in R' have a one-to-one correspondence with records in R. This is done by including the key column(s) of R in R'. When a query $Q \in \mathbf{W}$ is executed, for each record in R required to answer Q, we append the query id of Q to *TagColumn* of the corresponding record in R'. When R' is sorted by *TagColumn*, records belonging to the same fundamental region appear together. We experimentally evaluate the overhead of tagging in Section 9. We note that techniques reported in [11] can be used to further reduce the cost of tagging records. Also, for selection queries with aggregation, we can also use a bit vector representation for *TagColumn* (instead of varchar) where bit *i* is set if query Q_i requires this record to answer the query. However, this representation is not possible for queries with GROUP BY since the tag also needs to encode the identity of the group (see Section 8.1).

7.3.2 Handling Large Number of Fundamental Regions

To build the expression for MSE ($p_{\mathbf{W}}$), for each query Q in **W** the algorithm has to visit each fundamental region. If there are *q* queries in **W** and *r* fundamental regions, the product $q*r$ can become quite large. We handle this potential scalability issue by eliminating regions of low *importance* (defined in Section 4.3) immediately after they have been identified. For SUM queries, we used a similar technique, where the importance of each region is $f_i*Y_i^2$ where Y_i is the sum of the values of the aggregate column within the region. Our experiments show that this heuristic for pruning does not significantly affect quality.

7.3.3 Obtaining Integer Solutions

In Section 7.1.2 we presented a solution to the optimization problem in which the k_j's (number of records allocated to region R_j) could be fractional. In reality however, we are required to pick an integral number of records from each region during the sampling step. We observe that no previous work has addressed this important issue. In general if most of the k_j's are greater than 1, then the following simple rounding scheme works adequately. We round down each k_j to $\lfloor k_j \rfloor$. The leftover fractions are accumulated, and redistributed in a greedy manner to the regions that increase the MSE the least. We are also investigating randomized rounding schemes (as discussed in [14]).

7.3.4 Obtaining an Unbiased Estimator

If many k_j's are < 1, then after the rounding is performed the allocation algorithm may assign no samples to many regions. Moreover, fundamental regions that have been pruned out for scalability reasons will also not receive any samples. Due to both these reasons, we may get a *bias* in the estimates, i.e. the expected value of the answer may no longer be equal to the true answer. This issue can be addressed in a systematic manner by "merging" the fundamental regions with no allocated samples with the other fundamental regions such that the MSE is affected as little as possible. We omit these details due to lack of space.

7.4 Putting it All Together

Figure 4 summarizes the key steps in STRAT and analyzes their complexity. The tagging step (Step 1) is I/O bound and dominates the running time of STRAT in practice (see Section 9); its running time is dependent on the number of queries in the workload. Steps 2-3 identify the fundamental regions in the relation for the given workload **W** and can be accomplished in time $O(n*\log(n))$ where

n is the size of the relation. Thus, Steps 1-3 constitute the *stratification* step of STRAT. Steps 4-5 constitute the *allocation* step (which is CPU bound) and runs in time O(*q***h***u*), where *q* is the number of queries in **W**, *u* is the number of fundamental regions remaining after pruning. Finally, Step 6 is the sampling step that actually generates the sample(s) from the source relations, and can be done in once scan of each source relation.

1. For each query Q ∈ **W,** tag records in R used to answer query Q using the tagging algorithm described in Section 7.3.1
2. Let $R_1..R_u$ be the fundamental regions after pruning out unimportant fundamental regions (see Section 7.3.2).
3. For SUM queries, further divide each fundamental region R_j into *h* finer regions using the algorithm in Section 7.2.1.
4. For each query Q ∈ **W**, compute α_j of each (finer) region R_j referenced in Q, according to the formulas in Section 7.1.2 and 7.2.2. At the end of this step, we have computed an α_j for each R_j.
5. Solve the optimization problem of distributing ***k*** records to regions using the technique in Section 7.1.2. Let k_j be the number of records allocated to region R_j.
6. Perform stratified sampling to pick k_j records from region R_j and generate a sample of R.

Figure 4. Algorithm STRAT

8 EXTENSIONS FOR MORE GENERAL WORKLOADS

8.1 GROUP BY Queries

We first show how workloads containing GROUP BY queries can be lifted (see Section 5 for how a workload containing pure selection queries with aggregation can be lifted). Consider a GROUP BY query Q with weight *w* in the workload. Let Q partition R into *g* groups: G_1, … G_g. Within each group G_j, let S_j be the set of records selected. We adopt the following simple lifting model: replace Q in the workload with *g* separate selection queries with aggregation (each of weight *w*/g) that select S_1, …, S_g respectively, and use the techniques in Section 5 for lifting the resultant workload. The tagging step (see Section 7.3.1) logically treats each GROUP BY query Q as a collection of *g* selection queries with aggregation, and tags the records with the group that they belong to. During the tagging process, for GROUP BY columns of integer data types, we append a double <*c*, *v*> in addition to the query id to identify the group, where *c* is the column id of the GROUP BY column and *v* is the value of that column in record t. For non-integer data types, we treat the value of the GROUP BY column as a string and use a string hashing function to generate an integer value.

8.2 JOIN Queries

Our algorithm can be easily extended to a broad class of queries involving foreign key joins over multiple relations. In particular, we can handle workloads containing *star queries*, which are widely used in the context of the decision support. A star query is one that contains (a) one source relation and a set of dimension relations connected via foreign-key joins (b) GROUP BY and selections on source and/or dimension relations and (c) aggregation over columns of the source relation. Two approaches for handling star queries are possible, and our techniques apply to both. One approach is to identify a sample only over the source relation. For an incoming query, we can then join the sample over the source relation with the dimension relations in their entirety to compute the aggregate. This method is reasonable because typically the source relation is large (where sampling helps), while the dimension relations are relatively smaller. Another approach is to identify a sample of the source relation *and* precompute its join with all dimension relations (this is similar to join synopses presented in [2]). For an incoming query, we can avoid computing joins at runtime. For *both* approaches the allocation of samples can be done by setting up $MSE(p_{\mathbf{W}})$ and minimizing it. We note that for a given space constraint, the latter approach achieves faster query execution time at the expense of reduced accuracy. A detailed comparison of the two approaches is omitted due to lack of space.

Finally, observe that the tagging step (Section 7.3.1) for both approaches is similar to the single table query case. Of course, only records in the source table that satisfy all selection *and* join conditions in the query are tagged.

8.3 Other Extensions

Consider a workload containing a mix of COUNT and SUM queries. We need to make sure that each term MSE(p{Q}) is set up appropriately to reflect the type of query Q in the workload since, as explained above, analysis for COUNT and SUM differ. Once these expressions are set up, minimizing the resulting $MSE(p_W)$ is straightforward. Similarly, we can handle a mix of queries of the form SUM(*x*), SUM(*y*), SUM (<expression>) (e.g., SUM(*x***y*+*z*)), and other aggregates such as AVG (which is treated as SUM/COUNT). We can also extend our techniques to handle cases when the workload consists of aggregation queries with nested sub-queries, as also single-table selection queries with aggregation but where each query can potentially reference a different relation. Details of these extensions are omitted due to lack of space.

9 IMPLEMENTATION AND EXPERIMENTAL RESULTS

We have implemented **STRAT** and **FIXED** on Microsoft SQL Server 2000 and conducted experiments to evaluate their effectiveness. We compared their quality and performance with the following previous work: (a) uniform random sampling (**USAMP**) (b) weighted sampling (**WSAMP**) [6,11], (c) outlier indexing combined with weighted sampling (**OTLIDX**) [6], and (d) Congressional sampling (**CONG**) [1]. We describe the implementation of the previous work, our experimental setup, the results of the experiments, and draw conclusions.

9.1 Implementation

The key implementation aspects of FIXED and STRAT have been discussed in Sections 4, 7 and 8. We now briefly describe our grid search approach (Section 5) for automatically determining the appropriate values of δ and γ for a workload **W**. We divide the workload into two equal halves called the training and test set respectively. We divide the two-dimensional space $0.5 \le \delta \le 1$, $0 \le \gamma \le 0.5$ into a grid in which each dimension is divided into a fixed number of intervals. For each point (δ, γ) in the grid, we compute a sample for the training set and estimate the error for the test set. We pick the grid point with the lowest error for the

test set as our setting for δ and γ. Our implementation scales well with data size since we can obtain samples for multiple grid points in one scan of the relation.

We now briefly describe our implementation of the previous work. For uniform sampling (USAMP), each record is accepted with probability equal to the sampling fraction. We generate a uniform random sample in one scan of the relation R using the *reservoir sampling* technique [21]. For weighted sampling (WSAMP) [6,11] the probability of accepting a record is proportional to the frequency with which the record is selected by queries in the workload. We calculate this frequency for each record using the tagging technique described in Section 7.3.1. The key difference is that rather than keeping track of the list of queries that select the record, we only need a single counter (an integer) for the *TagColumn* to keep track of the frequency. For the outlier-indexing method (OTLIDX), we implemented the technique described in [6]. The paper does not address the following issue: for a given sample size, how many records of the sample to allocate for the outlier index, and how many to the weighted sample? To give OTLIDX the best possible choice of alternative settings, we tried different strategies for partitioning the sample for different databases and workloads – 25% for outliers-75% for weighted sample, 50%-50% and 75%-25%. We use the 50%-50% strategy since it performed well for most workloads. We also implemented the Congress algorithm described in the paper [1]. The algorithm takes as input a set G of GROUP BY columns and builds a sample for answering queries on any subsets of G (including ∅). For each subset of G, it determines the best allocation for each of the finest groups in the relation. The final allocation for a group is proportional to the maximum allocation for that group over all subsets of G. Since the algorithm for Congress that takes into account selections in the workload is not publicly available, in our experiments we only evaluate Congress for workloads consisting of pure GROUP BY queries (i.e., no selections).

9.2 Experimental Setup

Hardware/OS: All experiments were run on a machine having an x86 550 MHz processor with 256 MB RAM and an internal 18GB hard drive running Microsoft Windows 2000.

Databases: We used the popular TPC-R benchmark [20] for our experiments. One of the requirements of the benchmark however, is that the data is generated from a *uniform* distribution. Since we were interested in comparing the alternatives across different data distributions, we used the publicly available program [9] for generating TPC-R databases with differing data skew. For our experiments we generated 100MB TPC-R databases by varying the Zipfian [24] parameter *z* over values 1, 1.5, 2, 2.5, and 3. We report a few relevant characteristics of the data in the aggregation column used. First, the ratio of the maximum to the minimum value in the aggregation column varied between approximately 9000 and 250000 for the different databases. Second, there is no correlation between values in the aggregation column (picked from the Zipfian distribution) and their frequency in the data.

Workloads: We generated several workloads over the TPC-R schema using an automatic query generation program. The program has the following features that can be turned on: (i) aggregations on the fact table (*lineitem*), (ii) foreign-key joins between the fact table and a dimension table (*part* or *supplier*), (iii) grouping and (iv) selection. We experimented with three classes of workloads containing aggregation: (a) W-SEL (Selections, Foreign-Key Joins). (b) W-GB (Group By, Foreign-Key Joins) (c) W-SEL-GB (Selections, Group By, Foreign-Key Joins. Thus, e.g., W-SEL-GB-100 indicates a workload from the W-SEL-GB class containing 100 queries. The selection conditions were on the following columns*: l_shipdate, l_orderkey, l_tax, l_discount, p_partkey, p_size, p_retailprice, s_acctbal, s_suppkey*. As in [1], we used the grouping columns *l_shipdate, l_returnflag* and *l_linestatus*. The aggregate column was *l_extendedprice*, and the aggregation expressions used were COUNT and SUM. For each workload, we used the first half of the workload as the *training set* that was used to determine the sample, and the second half as the *test set*. We controlled the degree of similarity between the training and test set using the following two parameters: (a) The set of columns on which conditions are allowed in the training set and in the test set. (b) For each column on which a selection is defined, control the range of the selection condition.

Parameters: We varied the following parameters in our experiments: (a) Skew of the data, *z* (b) The sampling fraction *f* was varied between: 0.1% - 10%. (c) Workload size was varied between 25 - 800 queries. All numbers reported are the average over multiple runs.

Error Metric: As with previous work, we report the average relative error over all queries in the workload, i.e., L_1 metric (Section 3.3). We have found in our experiments that similar trends also hold for the RMSE (L_2) error metric (see Section 3.3).

9.3 Results

Quality vs. Sampling Fraction: We compare the quality (errors) of the various techniques for the COUNT and SUM aggregates as the sampling fraction is varied keeping the workload (W-SEL-GB-100) and data skew (z=2) fixed. As we see from Figures 5 and 6, for the *test set* (for COUNT and SUM aggregates respectively), the errors for STRAT are relatively low even with as little as 1% sampling whereas errors with other methods (USAMP, WSAMP, OTLIDX) are significantly higher. The key point to note for the SUM aggregate is that STRAT is able to achieve better quality than OTLIDX by taking into account the variance in the data values in a more principled way.

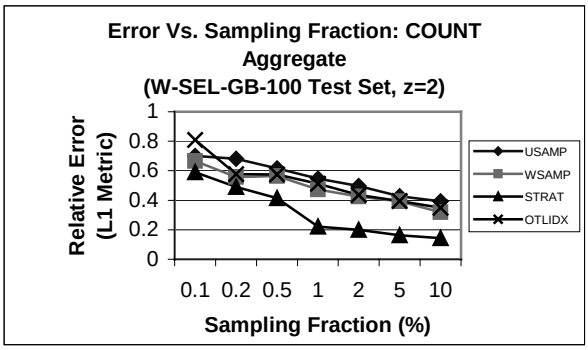

Figure 5. COUNT Aggregate – Test Set

Next, we compare the quality of the various alternatives for the *training set* itself. We see the effectiveness of our stratification algorithm from Figure 7 (for the COUNT aggregate), where STRAT gives errors close to 0 once the sample size exceeds the number of fundamental regions induced by the workload. For comparisons with CONG, we consider workloads with only GROUP BY queries (i.e., no selection). Figure 8 shows that for the COUNT aggregate, STRAT performs best among all methods.

We note that CONG also does significantly better than the other methods. The reason STRAT is more accurate than CONG is that despite attempting to account for all groups, CONG still allocates too many records to large groups and not enough for small groups, whereas STRAT is able to balance the allocations better.

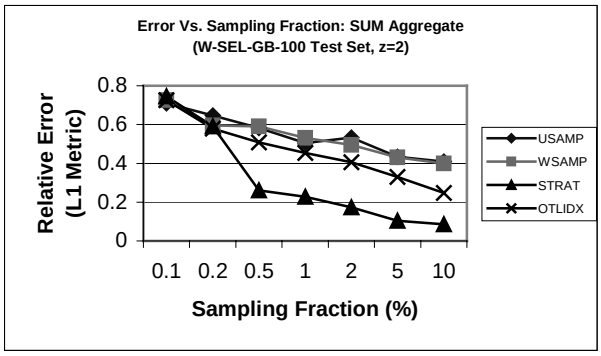

Figure 6. SUM Aggregate – Test Set

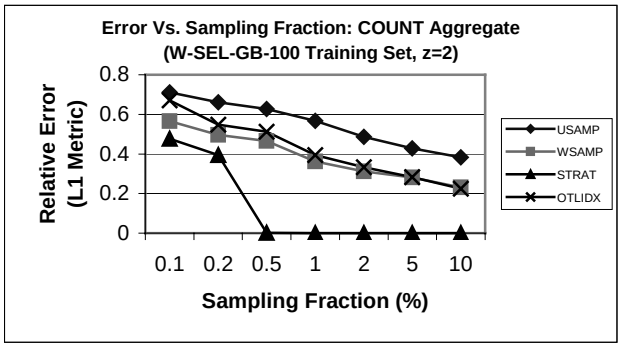

Figure 7. COUNT Aggregate – Training Set

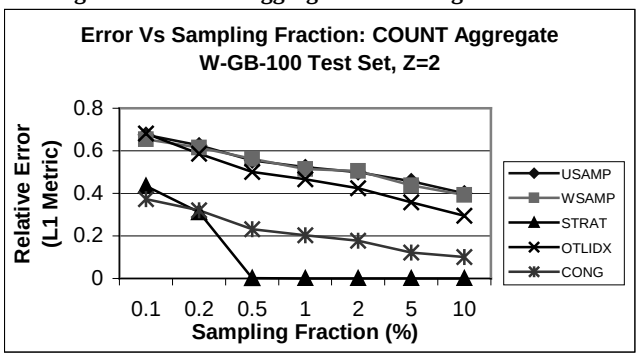

Figure 8. GROUP BY only workload. COUNT Aggregate – Test Set

Quality Vs. Overlap between Training set and Test set: We vary the degree of overlap of the minimum and maximum values of the range from which selection conditions are generated. The degree of overlap is an informal measure of *correlation*. For example, a degree of overlap of 0% (negative correlation) implies that for each column in a selection condition, the range of values from which selection conditions can be chosen for the test and training set for each column are disjoint, whereas 100% overlap (positive correlation) implies that the ranges are the same. From Figure 9 we see that for small overlap, as expected STRAT (δ = 0.90, γ = 0.01) gives higher errors than other methods. However, for moderate to large overlaps, STRAT is significantly better.

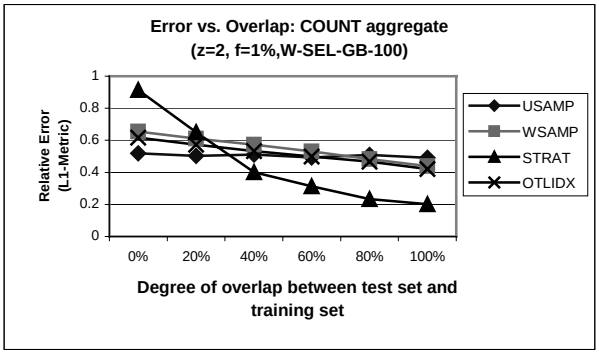

Figure 9. Varying overlap between training set and test set.

STRAT vs. FIXED: We compare STRAT (with δ = 0.99 and γ = 0.01) with FIXED for the given workload (i.e., on the training set) to illustrate the benefits of our deterministic solution. Note that the setting of δ and γ imply that we expect queries that are very similar to the given workload. We use the W-SEL-GB-100 workload, sampling fraction *f*=0.2%, and varied the data skew; we report errors for the COUNT aggregate. We found that across all data skew values, FIXED gives significantly lower error (difference in error varied between 13%-29%) since the deterministic method is able to exploit the greater freedom it has in optimizing the samples (see Section 4). We note that (a) unlike STRAT, FIXED has the drawback that we cannot report a standard error for the estimate, (b) for higher sampling fractions STRAT also approaches near-zero errors (Figures 7).

Quality vs. Data Skew: In this experiment we compared the quality of the different methods as the skew of the data (*z*) is varied between 1 and 3, keeping the workload (W-SEL-GB-100) and sampling fraction (1%) fixed, for the SUM aggregate. We found that for moderately skewed to highly skewed data ($z > 1$) , STRAT gives significantly lower errors than other methods (by about 20%). For low skew data (*z*=1), the other methods are comparable to STRAT.

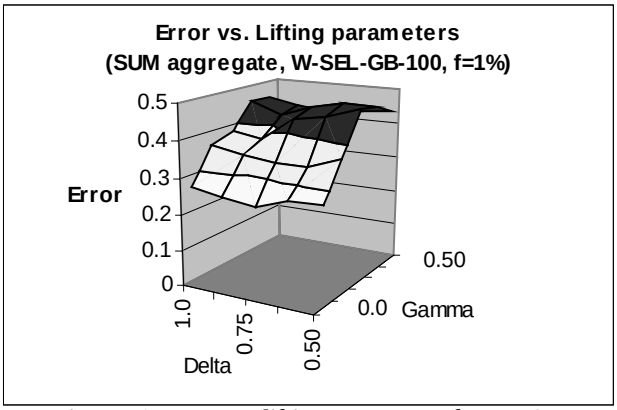

Figure 10. Error vs. lifting parameters for test Set.

Automatically determining the lifting parameters δ and γ: For a given workload W-SEL-GB-100 and sampling fraction of 1%, Figure 10 shows how the error for the test set varies with δ and γ (see Sections 5, 9.1). We see that the error varies gradually, which indicates that our grid search approach is promising.

Comparison on a real data set: We compare the quality of various approaches on a real data warehouse within our organization, used to track sales of products. We used δ = 0.90

and γ=0.01 for STRAT. We used a portion of the database of approximately 0.84 million rows; training and test sets of 25 real queries used by the application each. These queries typically contained 3-6 GROUP BY columns and 2-5 selection conditions per query. Figure 11 shows that for the test set, STRAT performs consistently better than other methods for this real data set.

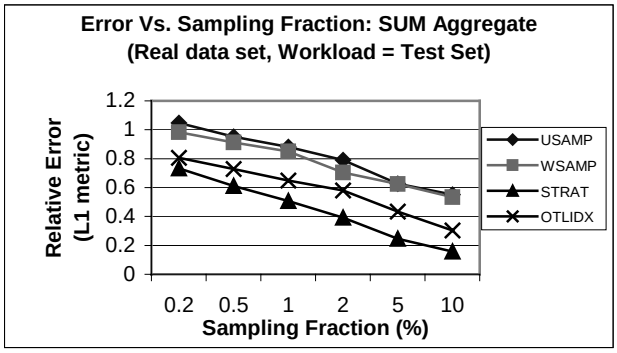

Figure 11. Error vs. Sampling Fraction.

Comparison of time for building samples: We compare the time to build the sample for WSAMP, OTLIDX, and STRAT for three different workloads of 100 queries each. We report numbers for the 100 MB database, data skew *z*=2. Figure 12 shows that the additional time (relative to WSAMP) taken by STRAT to tag the database (Section 7.3.1) for the given workload is small. The difference between the tagging for WSAMP and STRAT is that in STRAT we additionally need to record the query id information (and for GROUP BY queries, the group information). Finally, for a 1% sample, we report that the time to actually pick the sample after tagging was 15 sec, 70 sec, and 36 sec respectively for WSAMP, STRAT and OTLIDX for the W-SEL-GB-100 workload. Thus, the total time to build a sample is dominated by the time taken to tag the relation for the given workload.

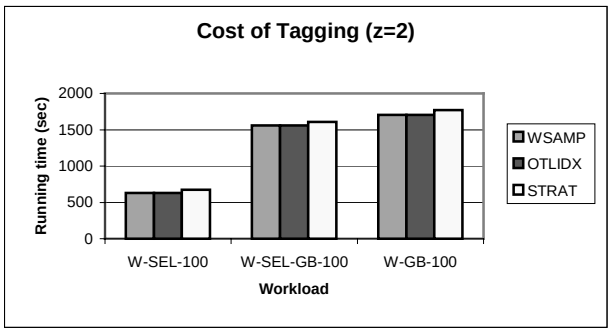

Figure 12. Comparison of running time to build sample.

10 SUMMARY

In this paper, we present a comprehensive solution to the problem of identifying samples for approximately answering aggregation queries, and show how it can be implemented on a database system. Using a novel technique for lifting a workload, we make our solution robust enough to work well even for workloads that are similar but not identical to the given workload. Our solution handles the problems of data variance, heterogeneous mixes of queries, GROUP BY and foreign-key joins.

11 ACKNOWLEDGMENTS

We are thankful to Venkatesh Ganti for his thoughtful comments on this paper.

12 REFERENCES

[1] Acharya S., Gibbons P.B., Poosala V. Congressional Samples for Approximate Answering of Group-By Queries. Proc. of ACM SIGMOD, 2000.

[2] Acharya S., Gibbons P.B., Poosala V., Ramaswamy S.. Join Synopses for Approximate Query Answering. Proc. of ACM SIGMOD, 1999.

[3] Barbará D., and Sullivan M. Quasi-Cubes. Exploiting Approximations in Multidimensional Databases. SIGMOD Record, Vol. 26 No. 3, Sep. 1997.

[4] Barbará D. and Wu. X. Using Approximations to Scale Exploratory Data Analysis in Datacubes. Proceedings of the 1999 ACM SIGKDD Int'l Conference on Knowledge Discovery and Data Mining, San Diego, CA, Aug. 1999.

[5] Chakrabarti K., Garofalakis M., Rastogi R., Shim K. Approximate Query Processing Using Wavelets. Proc. of VLDB 2000.

[6] Chaudhuri S., Das G., Datar M., Motwani R., Narasayya V. Overcoming Limitations of Sampling for Aggregation Queries. Proc. of IEEE Conf. on Data Engineering, 2001.

[7] Chaudhuri S., Das G., Narasayya V. A Robust, Optimization- Based Approach for Approximate Answering of Aggregation Queries. Microsoft Research Technical Report MSR-TR-2001-37.

[8] Chaudhuri S., Motwani R., Narasayya V. Random Sampling Over Joins. Proc. of ACM SIGMOD, 1999.

[9] Chaudhuri S., Narasayya V. Program for TPC-D Data Generation with Skew. http://research.microsoft.com/dmx/

[10] Cochran W.G. Sampling Techniques. John Wiley & Sons, New York, Third edition, 1977.

[11] Ganti V., Lee M.L., Ramakrishnan R.. ICICLES: Self-tuning Samples for Approximate Query Answering. Proc. of VLDB, 2000.

[12] Golub G., Loan C. Matrix Computations. Johns Hopkins University Press, 1989.

[13] Hellerstein J., Haas P., Wang H. Online Aggregation. Proc. of ACM SIGMOD,1997.

[14] HochBaum B. Approximation Algorithms for NP-Hard Problems. PWS Publishing, 1997.

[15] Ioannidis Y., Poosala V. Histogram Based Approximations of Set-Valued Query Answers. Proc. of VLDB 1999.

[16] Marron J.S, Smoothing Methods for Learning from Data. Proc. of KDD 1998, Tutorial.

[17] Poosala V., Ganti V. Fast Approximate Answers to Aggregate Queries on a Data Cube. Proc. of the 1999, Intl. Conf. on Scientific and Statistical Database Management.

[18] Silverman, B. W. Density Estimation. Chapman and Hall. 1986.

[19] Thisted R.A. Elements of Statistical Computing, Chapman and Hall. 1988.

[20] TPC Benchmark R. Decision Support. Revision 1.1.0. http://www.tpc.org.

[21] Vitter J. Random Sampling with a Reservoir. ACM Transactions on Mathematical Software, 11(1):37-57, March 1985.

[22] Vitter J., Wang M. Approximate Computation of Multidimensional Aggregates of Sparse Data using Wavelet. Proc. of ACM SIGMOD, 1999.

[23] Vitter J., Wang M., Iyer B.. Data Cube Approximation and Histogram via Wavelets. Conf. on Information and Knowledge Management, 1998.

[24] Zipf G.E.. Human Behavior and the Principle of Least Effort. Addison-Wesley Press, Inc, 1949.