Dynamic Sample Selection for Approximate Query Processing

Brian Babcock*
Stanford University
babcock@cs.stanford.edu

Surajit Chaudhuri
Microsoft Research
surajitc@microsoft.com

Gautam Das
Microsoft Research
gautamd@microsoft.com

ABSTRACT

In decision support applications, the ability to provide fast approximate answers to aggregation queries is desirable. One commonly-used technique for approximate query answering is sampling. For many aggregation queries, appropriately constructed biased (non-uniform) samples can provide more accurate approximations than a uniform sample. The optimal type of bias, however, varies from query to query. In this paper, we describe an approximate query processing technique that dynamically constructs an appropriately biased sample for each query by combining samples selected from a family of non-uniform samples that are constructed during a pre-processing phase. We show that dynamic selection of appropriate portions of previously constructed samples can provide more accurate approximate answers than static, non-adaptive usage of uniform or non-uniform samples.

1. INTRODUCTION

In recent years, advances in data collection and management technologies have led to a proliferation of very large databases. These large data repositories are typically created in the hope that through analysis, such as data mining and decision support, they will yield new insights into the data and the real-world processes that created it. In practice, however, while the collection and storage of massive data sets has become relatively straightforward, effective data analysis has proven more difficult to achieve. One reason that data analysis successes have proven elusive is that most analysis queries, by their nature, require aggregation or summarization of large portions of the data being analyzed. For multi-gigabyte data repositories, this means that processing even a single analysis query involves accessing enormous amounts of data, leading to prohibitively expensive running times. This severely limits the feasibility of many types of analysis applications, including those that require *timeliness* or *interactivity*.

*This work was performed while the author was visiting Microsoft Research.

SIGMOD 2003, June 9-12, San Diego, CA.

Ad hoc, exploratory data analysis is a cognitively demanding process that typically involves searching for patterns in the results of a series of queries in order to formulate and validate a hypothesis. This process is most effective when it can be performed interactively; long pauses between the time that a query is asked and the time when the answer is available are likely to be disruptive to the data exploration process. While keeping query response times short is very important in many data mining and decision support applications, exactness in query results is frequently less important. In many cases, "ballpark estimates" are adequate to provide the desired insights about the data, at least in preliminary phases of analysis. For example, knowing the marginal data distributions for each attribute up to 10% error will often be enough to identify top-selling products in a sales database or to determine the best attribute to use at the root of a decision tree.

The acceptability of inexact query answers coupled with the necessity for fast query response times has led researchers to investigate *approximate query processing* (AQP) techniques that sacrifice accuracy to improve running time, typically through some sort of lossy data compression. The New Jersey Data Reduction Report [6] provides an overview of many of the techniques that have been tried. The general rubric in which most approximate query processing systems operate is as follows: first, during the "pre-processing phase", some auxiliary data structures are built over the database; then, during the "runtime phase", queries are issued to the system and approximate query answers are quickly returned using the data structures built during the pre-processing phase.

The requirement for fast answers during the runtime phase means that scanning a large amount of data to answer a query is not possible, or else the running time would be unacceptably large. Thus, most approximate query processing schemes have restricted themselves to building only small auxiliary data structures such as a small sample of the data (e.g. a random subset of rows of the original database table). However, because relatively large running times and space usage during the pre-processing phase are generally acceptable as long as the time and space consumed are not exorbitant, nothing prevents us from *scanning* or *storing* significantly larger amounts of data during pre-processing than we are able to access at runtime. Of course, because we are only able to access a small amount of stored data at runtime, there is no gain to be had from building large auxiliary data structures unless they are accompanied by some indexing technique that allows us to decide, for a given query, which (small) portion of the data structures should be accessed to

produce the most accurate approximate query answer.

In this paper, we introduce a general system architecture for approximate query processing that is based on a technique that we call *dynamic sample selection*. The basic idea is to construct during the pre-processing phase a large number of differently biased samples, and then, for each query that arrives during the runtime phase, to dynamically select an appropriate small subset from the samples that can be used to give a highly accurate approximate answer to the query. The power of dynamic sample selection stems from the observation that, for most queries, an appropriately biased sample can produce more accurate approximate answers than a uniform sample. Previous attempts to exploit this observation via non-uniform sampling ([2, 10]) sample using a bias that is carefully chosen with the intent to provide good accuracy across a particular set of queries. However, what constitutes an "appropriate" bias can be quite different from one query to the next, so no single biased sample can be effective for all queries. In constrast to previous techniques which relied on a single sample with a fixed bias, dynamic sample selection constructs an individually tailored sample for each query in a semi-online fashion: the creation of the subsamples used as building blocks is performed offline but their assembly into an overall sample is done online.

The philosophy behind dynamic sample selection is to accept greater disk usage for summary structures than other sampling-based AQP methods in order to increase accuracy in query responses while holding query response time constant (or alternatively, to reduce query response time while holding accuracy constant). We believe that for many AQP applications, response time and accuracy are more important considerations than disk usage. For these applications, the trade-off chosen by dynamic sample selection is the right one.

Besides the generic dynamic sample selection architecture, a significant contribution of this paper is the development of a particular instantiation of dynamic sample selection that we call *small group sampling*. Small group sampling is designed to answer a standard class of analysis queries, aggregation queries with "group-bys". We give a detailed exposition of the small group sampling algorithm and study its performance through an extensive set of experiments on real and synthetic datasets. Our experimental comparisons demonstrate that small group sampling outperforms previously known approximate query processing techniques.

The organization of the remainder of the paper is as follows: First, in Section 2, we provide an overview of previous work in approximate query processing. Then, in Section 3, we describe the dynamic sample selection architecture in detail. In Section 4 we describe small group sampling and provide analytical justification for the approach, while in Section 5 we give experimental validation of its effectiveness. We conclude with a summary of our contributions in Section 6.

2. RELATED WORK

Some of the most studied methods for addressing the long running times of data analysis queries are not AQP techniques at all, but rather OLAP query processing techniques designed to more efficiently produce exact answers to analysis queries. Examples of this class of techniques include constructing materialized views of "data cubes" [17] over commonly-queried attributes as in [18, 20] and building indexes targeted at analysis queries [4, 14, 19]. These physical data design techniques typically make use of significant preprocessing time and space, but they are often quite effective at speeding up specific queries, particularly when the query workload is known in advance and can be leveraged during pre-processing. However, since it is prohibitively expensive to build indexes or materialized views sufficient to cover all possible queries, such techniques are of limited value for answering ad hoc analysis queries; inevitably there will be certain unanticipated queries that "fall through the cracks" and are not aided by physical design, particularly in exploratory data mining and decision support applications. Therefore application of physical database design technology does not eliminate the need for AQP technology; rather, the two are complementary.

The area of approximate answering of aggregate queries has been the subject of extensive research. Hellerstein et al. [22, 26] describe techniques for *online aggregation* in which approximate answers for queries are produced during early stages of query processing and gradually refined until all the data has been processed. The online aggregation approach has some compelling advantages. For example, it does not require pre-processing, and it allows progressive refinement of approximate answers at runtime. However, there are two important systems considerations that represent practical obstacles to the integration of online aggregation into conventional database systems. First, stored relations are frequently clustered by some attribute, so accessing tuples in a random order as required for online aggregation requires (slow) random disk accesses. Second, online aggregation necessitates significant changes to the query processor of the database system.

Due to the difficulty of purely online approaches to AQP, most research has focused on systems that make use of data structures built by pre-processing the database. Sophisticated data structures such as wavelets [8] and histograms [23] have been proposed as useful tools for AQP. Work in these areas is of great theoretical interest, but as with online aggregation, its practical impact is often limited by the extensive modifications to query processors and query optimizers that are often needed to make use of these technologies. Partly for this reason, sampling-based systems have in recent years been the most heavily studied type of AQP system. Sampling-based systems have the advantage that they can be implemented as a thin layer of middleware which re-writes queries to run against sample tables stored as ordinary relations in a standard, off-the-shelf database server.

The AQUA project at Bell Labs ([1, 2, 3]) developed a sampling-based system for approximate query answering. Techniques used in AQUA included *join synopses* [3], which allow approximate answers to be provided for certain types of join queries, and *congressional sampling* [2], discussed further in Section 4.1. The problem of sampling-based approximate answers for join queries was also addressed in [12], which includes several strong negative results showing that many join queries are infeasibile to approximate using unweighted sampling.

Besides congressional sampling, several other weighted sampling techniques have been proposed that outperform uniform random sampling for certain types of queries. The use of workload information to construct biased samples to optimize performance on queries drawn from a known work-

load was considered in [10]. Workload information was also used in [15] to construct "self-tuning" biased samples that adapt to the query workload. The paper [9] proposes a technique called *outlier indexing* for improving sampling-based approximations for aggregate queries when the attribute being aggregated has a skewed distribution. Our small group sampling technique is similar to outlier indexing in that both augment ordinary uniform random samples with a small number of carefully chosen additional tuples.

The sample selection architecture proposed in this paper is dynamic in the sense that the sample used to answer a particular query is assembled dynamically at the time that the query is issued, rather than using a static, precomputed sample. This is in contrast to two other classes of techniques that are sometimes termed "dynamic" in the literature: *incremental maintenance* techniques [16, 25] that efficiently update data structures in response to changes in data, and *adaptive query execution* strategies [21, 24] that modify execution plans for long-running queries in respose to changing conditions.

3. DYNAMIC SAMPLE SELECTION

In this section, we describe the dynamic sample selection architecture for approximate query processing. Standard sampling-based AQP strategies are not able to take advantage of extra disk space when it is available because increasing the size of a sample stored on disk increases the running time of a query executing against that sample. Dynamic sample selection gets around this problem by creating a large sample containing a family of differently biased subsamples during the pre-processing phase but only using a small portion of the sample to answer each query at runtime. Because there are many different subsamples with different biases available to choose from at runtime, the chances increase that one of them will be a "good fit" for any particular query that is issued. Because only a small portion of the overall sample is used in answering any given query, however, the query response time is kept low.

To see why biased sampling is useful, consider the following example:

EXAMPLE 3.1. *Consider a database consisting of 90 tuples with Product="Stereo" and 10 tuples with Product="TV". Let us compare two different ways of selecting a ten-tuple random sample:*

1. *Select 10% of the tuples uniformly, each with weight 10.*
2. *Select 0% of the Product tuples and 100% of the TV tuples, and give each TV tuple weight 1.*

The weight of a tuple is the inverse of the sampling rate that was used for the partition of the database from which that tuple was drawn. To answer a count query using a sample, one scales each sample tuple by its weight.

Consider the query asking for the count of tuples where Product is TV. The second sample will always give the exact answer. The first sample will give the right answer only if exactly one of the TV tuples was chosen for the sample; this occurs only with probability 0.41. With probability 0.59, the estimate is off by at least a factor of two.

As the example shows, when portions of the database that are more heavily represented in a biased sample match with the portions of the database that are selected in a query, that sample will give a better estimate for that query than a sample where the reverse is true. Other factors can also cause a biased sample to be a "good fit" for a query. For example, when a measure attribute being summed has a skewed data distribution, more accurate approximate answers can be obtained by allocating a disproportionately large sample share to outlier values of the distribution [9].

3.1 Pre-Processing Phase

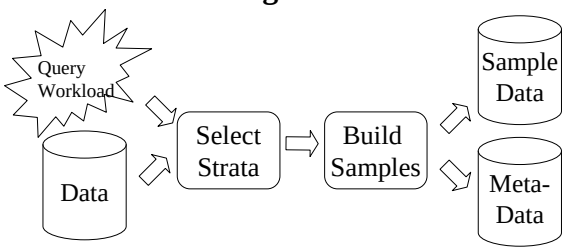


Figure 1: Pre-Processing Phase

In the dynamic sample selection architecture, the preprocessing of the database proceeds in two steps. In the first step, the data distribution of the database is examined to identify a set of biased samples to be created. The result of this step is a division of the database into (possibly overlapping) strata. If reliable query distribution information is available (e.g. a query workload), it can also be taken into account in this step. In the second step, the samples are created (potentially using a different sampling rate for each stratum) and stored in the database along with metadata that identifies the characteristics of each sample. This process is illustrated in Figure 1.

3.2 Runtime Phase

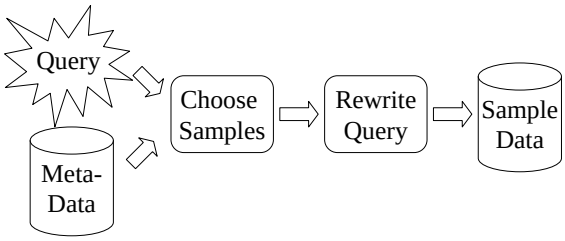


Figure 2: Runtime Phase

When queries are issued at runtime, the dynamic sample selection system re-writes the queries to run against sample tables rather than the original base tables referenced in the queries. The appropriate sample table(s) to use for a given query Q are determined by comparing Q with the metadata annotations for the samples. Figure 2 depicts this process.

The specific algorithms for choosing which samples are to be built during pre-processing and which samples are to be used for query answering during runtime are left unspecified in this abstract description of dynamic sample selection. The same generic system architecture can be used with a variety of different choices for these algorithms. Next, we give some examples of candidate algorithms that could be used.

3.3 Policies for Sample Selection

For dynamic sample selection to be successful, the samples that are created must be chosen in such a way that, when given a query, it is possible to quickly determine which of

the various samples to use to answer that query.

The simplest and most efficient class of dynamic sample selection strategies is one where the choice of samples is guided by the syntax of the incoming query. Some simple syntactic properties of queries have been used for sample selection in previous work. In [3], separate samples are created for each table in a database and the appropriate sample is chosen based on the table(s) referenced in the query's FROM clause, while in [9] a separate sample is created for use with each of a pre-specified list of aggregate expressions and the appropriate sample is chosen based on the aggregate expression in the query's SELECT clause.

The rest of this paper focuses on a powerful application of query syntax-based dynamic sample selection that we call *small group sampling*. In small group sampling, the sample tuples that are used for a query Q are determined by the grouping attributes from Q. We focus on small group sampling because it targets the most common type of analysis queries, aggregation queries with "group-bys". It also exemplifies the flexibility of dynamic sample selection because it selects from a large space of overlapping subsets of the precomputed sample when answering queries. In contrast, previous syntax-based schemes merely choose from a small number of non-overlapping partitions of the precomputed sample.

In principle, more complex policies for dynamic sample selection are possible that utilize information in addition to the query syntax. For example, in a workload-based AQP scheme, multiple samples may be created for workloads that differ from one another in characteristics that are not captured in syntax, e.g., overlap in the set of tuples accessed. In such a case, the choice of an appropriate sample for an incoming query may be determined by its relative "distance" from the samples for different workloads. Such distance computation can be based on analysis of query execution plans and workload-compression techniques discussed in [11]. Such techniques are likely to have higher overhead than syntax-based dynamic sample selection and will be studied in future work.

4. SMALL GROUP SAMPLING

Small group sampling is a specific dynamic sample selection technique that is designed for answering a common and useful class of queries: aggregation queries with "group-bys", the prototypical OLAP queries. In this section, we begin by describing the small group sampling technique in detail, then provide analytical justification for the technique.

We will consider the typical cases where (1) the queries are against a single fact table without any joins or (2) the queries are over a "star schema" where a fact table is joined to a number of dimension tables using foreign-key joins. We do not consider queries involving arbitrary joins because the results from [3, 12] demonstrate that random sampling approaches are futile for such queries; however, foreign-key joins represent the majority of joins in actual data analysis applications, so this restriction is not unduly limiting.

4.1 Motivation for Small Group Sampling

One of the shortcomings of uniform random sampling for answering group-by queries is that uniform samples give weight to each group in proportion to the number of tuples falling into that group. When the distribution of the sizes (measured in number of tuples) of the groups in the query is skewed, as is frequently the case in practice, this results in over-sampling of some groups and under-sampling of others. In contrast, the most accurate approximate answers to a group-by query are given when sample space is divided as equally as possible among the groups in the query (cf. Theorem 4.2 in [2]).

Unfortunately, the number of possible group-by aggregation queries is immense, so achieving the ideal sample allocation for each query by creating a separate sample per query is infeasible. Instead, what is required is some heuristic for choosing a set of samples that does a reasonable job on most group-by queries. Acharya, Gibbons, and Poosala [2] suggest such a heuristic, which they call *congressional sampling*. The basic idea behind congressional sampling is to consider the set of all possible group-by queries without any selection conditions and to calculate for each tuple t and each query Q the probability $p(t, Q)$ with which the tuple should be included in the optimal biased sample for Q, assuming some fixed sample size. Each tuple t is assigned a weight equal to $\max_{Q \in \mathcal{G}} p(t, Q)$ and a single stratified sample is constructed that includes each tuple with probability proportional to its weight. In essence, congressional sampling attempts to build a single sample that balances between all possible combinations of grouping columns.

In this paper, we propose the alternate *small group sampling* heuristic that improves on two shortcomings of congressional sampling. The first shortcoming is that since congressional sampling only creates a single sample, that sample must necessarily be very general-purpose in nature and only loosely appropriate for any particular query. Small group sampling uses the dynamic sample selection architecture and thus can realize the benefits of more specialized samples that are each tuned for a narrower, more specific class of queries. The second shortcoming is that the preprocessing time required by congressional sampling is proportional to the number of different combinations of grouping columns, which is exponential in the number of columns. This renders it impractical for typical data warehouses that have dozens or hundreds of potential grouping columns. In contrast, the pre-processing time for small group sampling is linear in the number of columns in the database.

The intuition behind small group sampling is that uniform sampling does a satisfactory job at providing good estimates for the larger groups in a group-by query since those groups will be well represented in the sample. It is the small groups that are the problem case for uniform sampling; however, precisely because the groups are small, it would not be excessively expensive to actually scan all the records contributing to small groups, assuming that we could identify them. The small group sampling approach uses a combination of a uniform random sample, which we call the *overall sample*, that provides estimates for large groups and one or more "sample" tables, which we call *small group tables*, that contain only rows from small groups. The small group tables are not downsampled — 100% of the rows from the small groups are included to ensure that the aggregate values for these groups can be given with complete accuracy.

Of course, the rows that fall into groups that are small will depend on the query that is asked. The set of groups in the query answer and their sizes depend on the grouping columns and the selection predicates of the query. The small group sampling heuristic builds tables containing the small groups from a specific set of aggregation queries: queries

with a single grouping column and no selection predicates. Each query's small groups are stored in a different table. This set of queries was chosen for several reasons:

- It is of manageable size, linear in the number of columns in the database.
- Determining which sample tables to use for any query Q is straightforward: besides the overall sample, the small groups tables for each grouping column in Q are queried, and a final approximate answer is composed out of the results of these queries.
- The tuples from small groups in a query that groups on a single column C and has no selection predicates will also be in small groups in all other queries (in the class that we consider) that include C in their group-by list.[1] Therefore the small group tables we build using single-column group-by queries without predicates will be broadly applicable to other group-by queries.

4.2 Description of Small Group Sampling

4.2.1 Pre-Processing Phase

The pre-processing algorithm for small group sampling takes two input parameters, the *base sampling rate* r, which determines the size of the uniform random sample that is created (i.e., the overall sample), and the *small group fraction* t, which determines the maximum size of each small group sample table. The parameters r and t are expressed as fractions of the total database size. For the purpose of understanding this section, "the database" means either the single fact table (for the single table schema) or the view resulting from joining the fact table to the dimension tables (for the star schema). We let N denote the number of rows in the database and $\mathcal{C}$ denote the set of columns in the database. The pre-processing produces three outputs: (a) an *overall sample table* with Nr rows; (b) a set of *small group tables*, one for each column in some set $\mathcal{S} \subseteq \mathcal{C}$ determined by the algorithm, with at most Nt rows in each table; and (c) a *metadata table* that lists the members of $\mathcal{S}$ and assigns a numeric index to each one. Such preprocessing can be implemented quite efficiently by making just two scans of the database. The purpose of the first scan is to identify the frequently occurring values for each column and their approximate frequencies. In the second scan, the small group tables for each column in $\mathcal{S}$ are constructed, along with the overall sample. The first scan may be omitted if sufficient information is already available in the database metadata, e.g. as histograms built for the query optimizer.

Initially, the set $\mathcal{S}$ is initialized to $\mathcal{C}$. In the first pass over the data, the small group sampling algorithm counts the number of occurrences of each distinct value in each column of the database in order to determine the common values for each column. This can be done using a separate hashtable for each column. For columns that have very large numbers of distinct values, the memory required to maintain such a hashtable could grow rather large. However, such columns are unlikely candidates to be grouping columns in the type of analysis queries that would be targeted at an AQP system, and furthermore small group sampling is not likely to be an effective strategy for such columns. Therefore, once the number of distinct values for a column exceeds a threshold τ (which we set to 5000 in our experiments), we remove that column from $\mathcal{S}$ and cease to maintain its counts. The memory required to maintain the hashtable of counts for each column is thus quite small. Since typical database columns have even fewer distinct values (e.g. dozens), the total memory required to simultaneously maintain the hashtables of all database columns is relatively modest.

After the first pass, the algorithm determines the set of common values $L(C)$ for each column C. $L(C)$ is defined as the mininum set of values from C whose frequencies sum to at least $N(1-t)$, and it is easily constructed by sorting the distinct values from C by frequency. Rows with values from the set $L(C)$ will not be included in the small group table for C, but rows with all other values will be; there are at most Nt such rows. It may be that a column C has no small groups, in which case it is removed from $\mathcal{S}$. After computing $L(C)$ for every $C \in \mathcal{S}$, the algorithm creates a metadata table which contains a mapping from each column name to an unique index between 0 and $|\mathcal{S}| - 1$.

The final step in pre-processing is to make a second scan of the the database to construct the sample tables. Each row containing an uncommon value for one or more columns (i.e. a value not in the set $L(C)$) is added to the small group sample table for the appropriate columns. At the same time as the small group tables are being constructed, the preprocessing algorithm also creates the overall sample, using reservoir sampling [28] to maintain a uniform random sample of rN tuples. Each row that is added to either a small group table or the overall sample is tagged with an extra bitmask field (of length $|\mathcal{S}|$) indicating the set of small group tables to which that row was added. This field is used during runtime query processing to avoid double-counting rows that are assigned to multiple sample tables.

Although the overall sample is described in the preceding paragraph as being a uniform random sample, it is also possible to use a non-uniform sampling technique to construct the overall sample; for example, in one of the experiments that we conduct in Section 5, we use outlier indexing [9] to construct the overall sample. In this respect, the small group sampling technique is orthogonal to other weighted sampling techniques and can be used in conjunction with them. For the remainder of this paper, the term "small group sampling" will refer to small group sampling with a uniform overall sample unless explicitly stated otherwise; e.g., we will refer to the aforementioned variant as "small group sampling enhanced with outlier indexing".

4.2.2 Runtime Phase

When a query arrives at runtime, it is re-written to run against the sample tables instead of the base fact table. Each query is executed against the overall sample, scaling the aggregate values by the inverse of the sampling rate r. In addition, for each column $C \in \mathcal{S}$ in the query's group-by list, the query is executed against that column's small group table. The aggregate values are unscaled when executing against the small group sample tables. Finally, the results from the various sample queries are aggregated together into a single approximate query answer.

Since a row can be included in multiple sample tables, the re-written queries include filters that avoid double-counting

[1] For COUNT and SUM, the aggregation functions that we consider in this paper, "smallness" is a monotonic condition in the number of grouping columns and in the number of selection predicates, meaning that if a group g is small in some query Q, then it remains small even if Q is modified by adding more grouping columns or additional selection predicates.

rows. When the query is re-written to run against the first small group table, no additional filtering is applied. When running against the second small group table, rows that were already included in the first small group table are filtered out, and so on. The rows to be filtered can be efficiently identified by applying an appropriate mask to the bitmask field that is present in all the sample tables. For example, consider the following single-table database query: `SELECT A, C, COUNT(*) AS cnt FROM T GROUP BY A, C`. Assume that small group preprocessing has been completed with a base sampling rate of 1%, and that small group tables exist for both column A and column C, and that these columns are assigned the indexes 0 and 2, respectively. Then the re-written query looks like:

```
SELECT A, C, COUNT(*) AS cnt FROM s_A
GROUP BY A, C
UNION ALL
SELECT A, C, COUNT(*) AS cnt FROM s_C
WHERE bitmask & 1 = 0
GROUP BY A, C
UNION ALL
SELECT A, C, COUNT(*) * 100 AS cnt
FROM s_overall
WHERE bitmask & 5 = 0  /*Since 5 = 2^0 + 2^2*/
GROUP BY A, C
```

To aid the user in interpreting the reliability of the approximate query answer, we also compute confidence intervals (not shown in above example) for each aggregate value in the query answer. Answers for groups that result from querying small group tables are marked as being exact, and confidence intervals for the other groups are provided using standard statistical methods, e.g. [5, 7]. Note that confidence interval calculation is very simple when using small group sampling because the source of inaccuracy can be restricted to a single stratum. In contrast, other stratified sampling techniques need to perform complex calculations involving the sampling rates for various strata to provide accurate confidence intervals.

4.2.3 Variations

The small group sampling technique admits several variations and extensions beyond the basic algorithm described. We briefly discuss a few variations as possible directions for future work. As an alternative to using single-column group-by queries, one could generate small group tables based on selected group-by queries over pairs of columns, or based on other more complex queries. The number of pairs of columns for an m-column database is $m(m-1)/2$, however, so some judgment would have to be exercised in selecting a small subset of pairs when m is large. Query workload information could also be used to trim the set of columns for which small group tables are built by identifying rarely-queried columns.

Small group sampling creates a two-level hierarchy: small groups are sampled at a 100% rate, while large groups are sampled at the base sampling rate r. This approach could be extended to a multi-level hierarchy. For example, one could sample 100% of rows from small groups, 10% of rows from "medium-sized" groups, and 1% of rows from large groups.

More sophistication could be added to the runtime selection of which small group samples to use. For example, for queries with a large number of grouping columns, using all relevant small group tables might result in unacceptably large query execution times; in this case, a heuristic for picking a subset of the relevant small group tables to query could improve performance.

Having described small group sampling, we now proceed with an evaluation of its effectiveness. We will first define the accuracy metrics we will use in our evaluation, then describe the results of analytical modeling, and complete the evaluation by giving experimental results.

4.3 Accuracy Metrics

Following [2], we consider two different accuracy criteria for approximate answers to group-by queries. First, as many of the groups as possible that are present in the exact answer should be preserved in the approximate answer. Second, the error in the aggregate value for each group should be small. In order to formalize these accuracy criteria as measurable error metrics, we introduce some notation. Given an aggregation query Q, let $G = \{g_1 \ldots g_n\}$ be the set of n groups in the answer to Q, and let x_i denote the aggregate value for group g_i. In the special case where Q is a simple aggregation without grouping, $n = 1$. Consider an approximate answer A for Q consisting of a set of m groups $G' = \{g_{i_1} \ldots g_{i_m}\}$ with aggregate values $x'_{i_1} \ldots x'_{i_m}$. Since we are concerned in this paper with sampling-based estimators, which never introduce spurious groups into the answer, we assume that $G' \subseteq G$.

DEFINITION 4.1. *The percentage of groups from Q missed by A is defined as:*

$$PctGroups(Q, A) = \frac{n - m}{n} \times 100$$

DEFINITION 4.2. *The average relative error on Q of A is defined as:*

$$RelErr(Q, A) = \frac{1}{n}\left((n - m) + \sum_{j=1}^{m} \frac{|x_{i_j} - x'_{i_j}|}{x_{i_j}}\right)$$

In other words, to compute the average relative error on Q of A, take the average relative error in the aggregate value, averaging across the groups in the exact answer and taking the relative error for each of the $n - m$ groups omitted from the approximate answer A to be 100%.

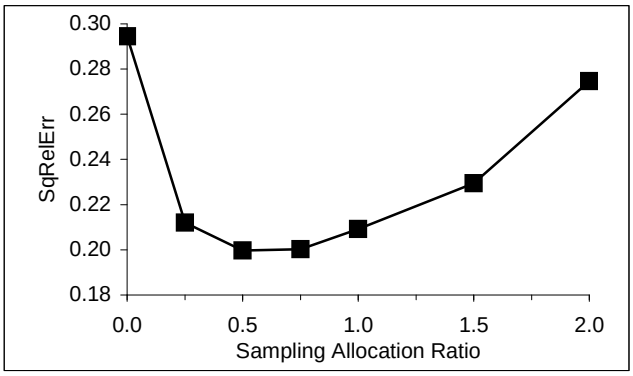
For analytical convenience, we define one additional metric, SqRelErr(Q,A), the average squared relative error on Q of A. We use SqRelErr in place of RelErr in our analytical comparison of uniform random sampling with small group sampling because it measures the same general objective (errors in aggregate values should be small for all groups) and is much more analytically tractable.

DEFINITION 4.3. *The average squared relative error on Q of A is defined as:*

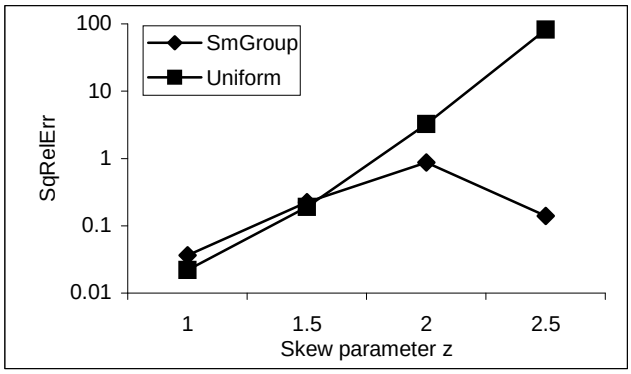
$$SqRelErr(Q, A) = \frac{1}{n}\left((n - m) + \sum_{j=1}^{m} \left(\frac{x_{i_j} - x'_{i_j}}{x_{i_j}}\right)^2\right)$$

4.4 Analysis

To quantify the benefits that we might expect to achieve from small group sampling, we conduct an analytical comparison of the expected performance of small group sampling and uniform random sampling on count queries over an idealized database. As mentioned in Section 4.3, for analytical convenience we use the SqRelErr metric instead of the RelErr metric. We also make the simplifying assumption that



(a) SqRelErr vs. Sampling Allocation Ratio



(b) SqRelErr vs. Skew

Figure 3: Analytical model of small group sampling

Bernoulli sampling is performed, so each tuple is independently included in the sample with probability p. (In actuality, we produce a fixed-size sample that has a p fraction of the overall rows.)

We first derive equations for SqRelErr for uniform random sampling and small group sampling. Given a count query Q over a database with N tuples, let $G = \{g_1 \ldots g_n\}$ be the set of n groups in the answer, and let p_i denote the fraction of the N tuples that belong to group g_i. Let $\mathcal{C}$ denote the set of grouping columns in the query Q and let $[v_{C,g_i} \in L(C)]$ denote the indicator function that equals 1 when the value for grouping column C in group g_i is one of the common values $L(C)$ and 0 otherwise. Consider A_u, an approximate answer for Q produced using uniform random sampling at sampling rate s/N, and A_{sg}, an approximate answer for Q produced using small group sampling with an overall sample generated at sampling rate s'/N. Let $E_u = E[SqRelErr(Q, A_u)]$ and $E_{sg} = E[SqRelErr(Q, A_{sg})]$ denote the expected values of the average squared relative error on Q of A_u and of A_{sg}, respectively.

THEOREM 4.1.

$$E_u = \frac{1}{sn} \sum_{g_i \in G} \frac{1-p_i}{p_i} \quad (1)$$

$$E_{sg} = \frac{1}{s'n} \sum_{g_i \in G} \left(\frac{1-p_i}{p_i} \prod_{C \in \mathcal{C}} [v_{C,g_i} \in L(C)] \right) \quad (2)$$

PROOF. Consider a particular group g_i in the answer to Q. If a uniform random sample S is created by including each tuple with probability s/N, then the number of tuples S_i from group g_i that are included in the sample will be binomially distributed with mean sp_i and variance $sp_i(1-p_i)$. To estimate the number of tuples in group g_i using the sample S, we scale S_i by the inverse sampling rate N/s. The resulting random variable has mean Np_i and variance $\frac{N^2 p_i(1-p_i)}{s}$. The squared relative error is equal to the squared error divided by the square of the actual group count Np_i, and the expected squared error is just the variance, so the expected squared relative error in the estimate of the count for group g_i is equal to $\frac{1-p_i}{sp_i}$. To compute E_u, then, we take the average of $\frac{1-p_i}{sp_i}$ over all groups $g_i \in G$, giving Equation 1.

When small group sampling is used, for those groups that are captured by the small group sample tables, there will be no error whatsoever because no downsampling is performed when constructing these tables. For all other groups, the expected squared relative error will be $\frac{1-p_i}{s'p_i}$. Averaging over all groups yields Equation 2. □

To ensure a fair comparision between different AQP systems, we allow each system to use the same amount of sample space per query at runtime. If small group sampling and uniform random sampling are both allowed to query s sample rows at runtime, then the size s' of the overall sample queried by small group sampling will be less than s since some of the s rows will come from small group tables. Small group sampling will be perfectly accurate on the groups covered by small group tables, but since $s' < s$, small group sampling will make somewhat larger errors than uniform random sampling on the other groups. Whether small group sampling will be preferrable to uniform random sampling depends on whether its precise accuracy on small groups compensates for its increased error on large groups.

The values of E_u and E_{sg} depend on the data distribution, the queries, and the allocation of available sample space between the overall sample and the small group tables. In order to understand when small group sampling excels and when it does poorly, we applied Equations 1 and 2 to a number of different query scenarios. Because Equations 1 and 2 are not in closed form, we computed the summations using a computer program and plotted the results graphically.

We assume that attributes are distributed according to a (truncated) Zipfian distribution, i.e. the frequency of the ith most common value for an attribute is proportional to i^{-z} for some constant z (called the *skew parameter*), except that the frequency is 0 if $i > c$ for some constant c that regulates the number of distinct attribute values. (We also assume for simplicity that the attributes are independent of one another.) To analyze the effects of varying data distribution, we tried out different values for z and c. To understand the effects of varying types of queries, we considered differing numbers of grouping columns g and selection predicates selectivities σ, assuming that a predicate of selectivity σ includes each tuple independently with probability σ. We also varied the *sampling allocation ratio* $\gamma = t/r$. (Recall that t and r control the sizes of each small group tables and the overall sample, respectively.)

We sought to answer the questions (1) what is the optimal relationship between the small group sampling's input parameters t and r, and (2) what is the relative performance of small group sampling and uniform random sampling under

various circumstances? Some of the results of our analytical simulations are shown in Figures 3(a) and 3(b).

Figure 3(a) shows the effect of various choices for the sampling allocation ratio. Uniform random sampling is equivalent to small group sampling with a sampling allocation ratio of zero. Our analysis suggests that a sampling allocation ratio of 0.5 performs well across a wide range of data distributions; this is the ratio that we used in the experiments described in Section 5. (The results illustrated in the figure are for $g = 2$, $\sigma = 0.1$, $c = 50$, and $z = 1.8$. Results for other values of these parameters were similar.) A sampling allocation ratio of 0.5 means that the maximum size for a small group sample table is half the size of the uniform random sample. As can be seen from the figure, however, the exact choice of the sampling allocation ratio is not critical, as values from 0.25 through 1.0 had similar results.

Figure 3(b) compares the performance of small group sampling with uniform sampling across a range of values for the skew parameter z. (We found that the query selectivity, number of grouping columns, and number of distinct values did not have a significant impact on the relative performance of the two strategies. This figure uses $g = 3$, $\sigma = 0.3$, $c = 50$, and $\gamma = 0.5$.) We found that, under our analytical model, uniform sampling is slightly preferrable to small group sampling for data that is uniformly distributed or very nearly so. For data distributions with moderate to high skew, our model showed small group sampling to be clearly superior to uniform random sampling.

Our analytical model is based on a number of simplifying assumptions that are unlikely to hold in real applications, but the results of our analysis provided some reason to believe that small group sampling might prove effective and led us to implement an AQP system to evaluate small group sampling empirically. We next describe the results of our experiments.

5. EXPERIMENTS

We implemented an AQP system using small group sampling and conducted a number of experiments to evaluate its accuracy and performance. We also implemented several previous AQP techniques such as (a) uniform random sampling, (b) congressional sampling [2], and (c) outlier indexing [9], and compared the accuracy and performance of small group sampling against these algorithms.[2] All algorithms were implemented as middleware that executed queries against a standard commercial database management system runnning on a back-end server (512Mhz Pentium processor with 256MB RAM).

5.1 Summary of Results

Accuracy Results

- For COUNT queries, the accuracy of small group sampling is significantly better than uniform random sampling and congressional sampling (basic congress).
- For COUNT queries, the accuracy of all methods degrade with (a) increasing number of grouping columns referenced in the query, (b) decreasing average group size of the query result, and (c) decreasing data skew. However, the degradation is less pronounced for small group sampling compared to uniform random sampling and basic congress.
- For SUM queries, the accuracy of small group sampling (enhanced with outlier indexing techniques) is better than outlier indexing techniques alone.

Performance Results

- The query processing times of all AQP algorithms are comparable to each other and orders of magnitude faster than executing the exact query; the speedup depends on the sampling rate.
- Small group sampling requires more space to store sample tables than the other algorithms. However, (a) unlike the other algorithms, small group sampling can gracefully take advantage of extra available space, and (b) the extra space is an acceptably small fraction of the space consumed by the original database tables.
- Small group sampling has acceptable pre-processing time; it is slower than uniform random sampling and outlier indexing, but comparable to basic congress.

In the rest of this section we discuss details of our experiments. We begin by describing our experimental setup, then present our results and draw conclusions.

5.2 Experimental Setup

5.2.1 Databases

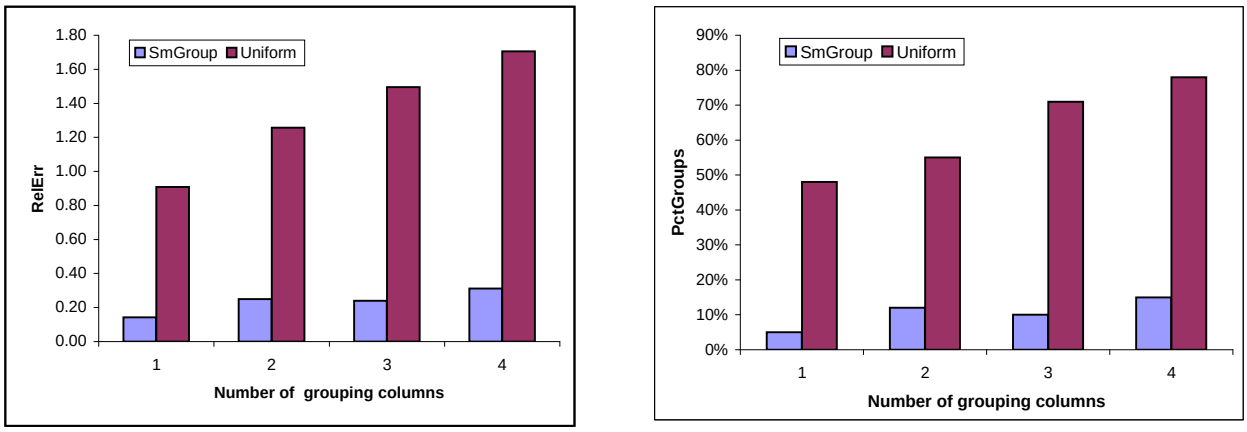
We used one real database and several synthetic databases for our experiments. The real-world database, which we will refer to as `SALES`, was a portion of a large corporate sales database. The database had a star schema with a fact table containing about 800,000 rows and 6 dimension tables, with the largest containing about 200,000 rows. The total number of columns in the fact and dimension tables considered for our experiments was 245. The database consumed approximately 1GB of disk space.

We also generated a series of synthetic databases based on the TPC-H benchmark [27]. We used a modified version [13] of the benchmark data generation program. The program was altered to produce skewed data according to a Zipfian distribution instead of the uniform distribution in the benchmark specification, so as to more accurately model real-world databases. We adopt a naming convention where `TPCHxGyz` refers to a database that was generated with scaling factor `x` and and Zipf parameter $z =$ `y` (thus for e.g., `TPCH1G2.0z` refers to a database with scaling factor 1 and Zipf parameter $z = 2.0$). We experimented with scaling factors 1 and 5 (which produced 1GB and 5GB databases, respectively) and z parameters of 1.0, 1.5, 2.0 and 2.5.

5.2.2 Sample Tables

In addition to the original tables, the various AQP algorithms require the creation of sample tables. To improve query processing performance, we joined each sample table with the dimension tables to produce *join synopses* as described in [3]. Because of the large number of sample tables used in small group sampling, we applied two space-saving techniques to the join synopses for the small group samples. First, we used the technique suggested in [3] of

[2]Note that we implemented a version of congressional sampling called *basic congress*; the more sophisticated *congress* algorithm did not scale for our experimental databases. Also, we do not present comparisons again other sampling-based AQP systems such as [10, 15] as these methods require the presence of workloads.



(a) RelErr vs. Number of grouping columns

(b) PctGroups vs. Number of grouping columns

Figure 4: Small Group vs. Uniform on TPCH1G2.0z

re-normalizing the join synopses back into their component tables; this produced a smaller version of each dimension table for each small group sample table. Second, we combined the resulting small dimension tables from all the small group sampling join synopses to create a single smaller dimension table for each of the original dimension tables.

5.2.3 Queries

For our experiments, we generated random select-project-join queries with group-bys and COUNT/SUM aggregations. The joins were restricted to foreign-key joins of the fact and dimension tables. We varied the number of grouping columns (between 1 and 4), the number of selection predicates(1 or 2), and the selectivity of the selection predicates. Grouping columns were chosen uniformly at random from the set of all columns in the database, except that columns where almost every value was unique (such as customer address) were excluded. Selection predicates were generated by choosing a column at random and then restricting to rows whose values for that column were from a randomly-chosen subset of the distinct values taken on by that column. We varied the size of this randomly-chosen subset between 0.05 and 0.3 times the number of distinct values for the column. For queries with multiple selection predicates, the WHERE clause included the conjunction of all predicates. For SUM queries the column being aggregated was randomly selected from specific measure columns.

For each combination of choices for the above variables, we generated 20 queries at random, thus generating a fairly large set of queries in total. We executed these queries using various AQP techniques and averaged the running time as well as the accuracy using the RelErr and PctGroups (see Section 4.3) error metrics.[3] We omit presenting experimental results using the SqRelErr metric as it was mainly used for analytical convenience; we observed it to to follow similar trends as the RelErr metric in our experiments. To ensure a fair comparision of running time and accuracy, we allowed each AQP technique to utilize the same amount of sample table space per query at runtime.

[3]Although we use TPC-H databases in our experiments, the queries used in our experiments are quite different from standard TPC-H benchmarks. Thus, our results do not reflect TPC-H benchmark numbers.

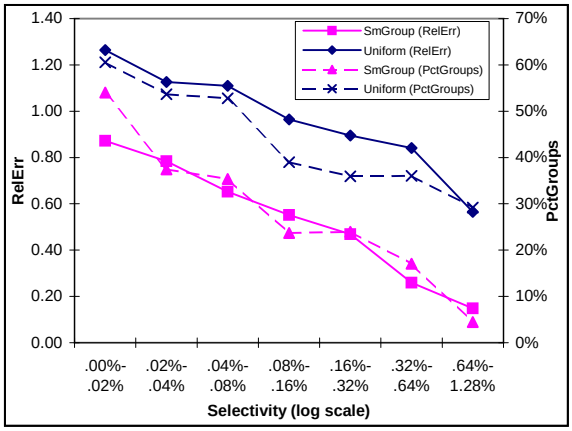


Figure 5: SmGroup vs. Uniform on SALES

5.3 Accuracy Experiments

5.3.1 Small Group vs. Uniform

We compared small group sampling with uniform random sampling on all databases. For small group sampling, we used a sampling allocation ratio of 0.5, as suggested by the analysis in Section 4.4. We experimented with various different sample sizes, but describe in detail results using a base sampling rate of 1%; i.e. where the size of the small group samplings's overall sample is 1% of the fact table and the size of each small group table is 0.5%. We measured the average RelErr and PctGroups error metrics over a large set of COUNT queries generated as described in Section 5.2.3. We allowed each technique to use the same amount of sample space during query execution; thus a query with i grouping columns executed using small group sampling is also executed on a uniform random sample of size $(1 + 0.5i)\%$.

We first present accuracy results on `TPCH1G2.0z`. Figure 4 shows how the error metrics vary as a function of the number of grouping columns referenced in the queries. For both methods, both RelErr and PctGroups increased as the number of grouping columns increased; the increase was more pronounced for uniform sampling than for small group sampling. For example, even for queries with 4 grouping columns, the average fraction of the groups from the exact

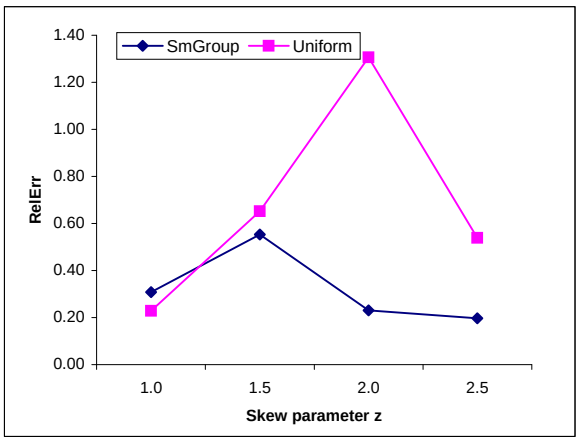


Figure 6: RelErr vs. Skew on TPCH1G databases

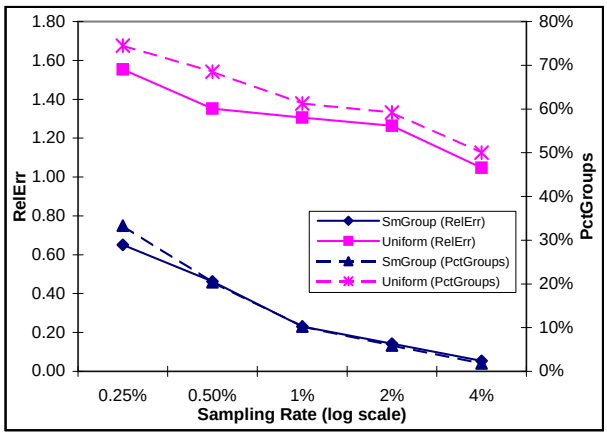


Figure 7: Error vs. Sampling Rate, TPCH1G2.0z

answer that were missed by small group sampling remained under 15%. Uniform sampling, on the other hand, missed more than 75% of groups per query for such queries.

We also measured how the error metrics vary as a function of the *per group selectivity* of the queries. The per group selectivity of a query is defined as the average group size (number of tuples in a group) in the query result. Intuitively, if the per group selectivity of a query is small, it has many small groups and is consequently harder to approximate using sampling methods. We varied per group selectivity from 0.02% to 1.28% and observed that while the accuracy of both AQP methods under both metrics increases with increasing per group selectivity, small group sampling is consistently much more accurate than uniform random sampling for this range of selectivities. For example, at 0.16% per group selectivity, RelErr for small group sampling was only 0.17, while for uniform sampling it was 1.23. At higher selectivities the differences were less pronounced as the resulting groups were large enough for uniform random sampling to do a competent job in estimating them accurately. Due to lack of space we omit including a chart to illustrate these results in detail.

We next present results of similar experiments on the real-world database, `SALES`. Figure 5 shows how the error metrics vary as a function of the per group selectivity (shown in log scale) of the queries. It can be seen that small group sampling is consistently better than uniform random sampling over the entire range of selectivities considered. For selectivities much larger than the range shown, the difference was less pronounced. We also measured how the error metrics vary as a function of the number of grouping columns referenced in the queries. Due to lack of space we omit including a chart to illustrate these results in detail, but mention that the metrics followed similar trends as in Figure 8 (discussed below in Section 5.3.2). The overall errors for both methods were somewhat higher compared to `TPCH1G2.0z`; this is perhaps attributable to the fact that the `SALES` database is relatively less skewed than the `TPCH1G2.0z` database.

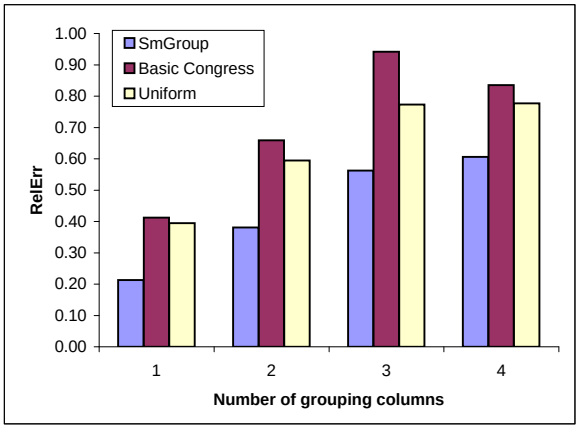
To systematically measure how the the accuracy of these AQP techniques are affected by data skew, we ran the above experiments on a series of `TPCH1Gyz` databases with varying skew. Figure 6 shows how the RelErr metric varies as a function of the Zipf parameter z. (The trends for PctGroups were almost identical.) The results are similar to the rule suggested by the analysis of Section 4.4: uniform sampling slightly outperforms small group sampling at low skew, while small group sampling does significantly better at moderate to high skew. The fact that the accuracy of uniform sampling improves somewhat at very high skew can be explained by noting that most attribute values occur very rarely in highly skewed data, so selection predicates often filter those values out altogether, leaving predominantly large groups. For skews in the range 1.5–2.0, which are consistent with the oft-cited 90-10 or 80-20 rules of thumb, small group sampling performs very well.

We conclude this subsection by briefly describing results when we vary the base sampling rate. For the `TPCH1G2.0z` database, we generated sample tables by varying the base sampling rate from 0.25% to 4%. The resuls are illustrated in Figure 7. We found that both RelErr and PctGroups for small group sampling and uniform random sampling degrade smoothly as the sampling rate is decreased. However, the accuracy of small group sampling was consistently better than uniform random sampling for all sampling rates.

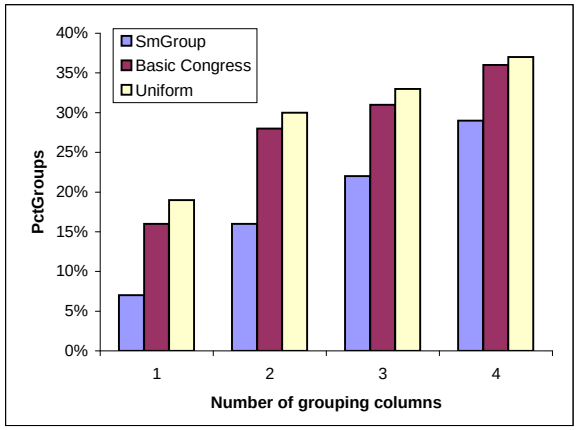
5.3.2 Small Group vs. Congress

We were unable to experiment with the *Congress* algorithm [2] on the `SALES` database because its running time is proportional to the number of possible combinations of group-by columns, and our database had 245 potential group-by columns, giving 2^{245} combinations. Instead, we implemented a more tractable version of the algorithm called *Basic Congress* (also described in [2]). We picked four dimension tables plus the fact table from the `SALES` database, and used their associated columns (120 columns in all) to build sample tables for small group sampling, basic congress, and uniform sampling, using a 1% base sampling rate.

Figure 8 shows how the error metrics vary as a function of the number of grouping columns referenced in the queries. Note that for comparision we have also included uniform random sampling in these charts. For all methods, both error metrics increased as the number of grouping columns increased; however, small group sampling was significantly more accurate than the other methods, whose accuracies were comparable to each other. Due to the large number of grouping columns considered, basic congress stratified the fact table into a large number (about 166,000) of tiny strata, and its sample table almost resembled a sample from a uniform distribution. This may explain why basic congress and uniform sampling performed similarly.



(a) RelErr vs. Number of grouping columns



(b) PctGroups vs. Number of grouping columns

Figure 8: Small Group vs. Basic Congress vs. Uniform on SALES

5.3.3 *Small Group vs. Outlier Indexing*

We used the `SALES` database to compare the performance of small group sampling enhanced with outlier indexing (see Section 4.2.1) with the outlier indexing algorithm alone [9]. For both methods, we generated sample tables with a base sampling rate of 1% and compared their accuracy over a large number of randomly generated SUM queries. The overall average RelErr metric was 0.79 for small group sampling enhanced with outlier indexing and 1.08 for outlier indexing alone. The overall average number of missed groups was 37% for small group sampling enhanced with outlier indexing and 55% for outlier indexing alone. These experiments demonstrated that for answering SUM queries, small group sampling enhanced with outlier indexing is consistently better than outlier indexing alone. We also compared uniform sampling to both these methods, and its accuracy was comparable to plain outlier indexing, with slightly higher RelErr and slightly lower PctGroups.

5.4 Performance Experiments

5.4.1 *Query Processing Performance*

We experimented with query processing performance of all AQP methods (using a 1% base sampling rate) on the 5GB databases (`TPCH5Gyz`); the other databases were smaller and hence the timing measurements were less reliable.

Since the total sample space given to all AQP algorithms was the same, their processing times were very similar. The overall average speedup of small group sampling was 9.49, i.e. approximately 10x faster than executing the query exactly. The overall average speedup of uniform sampling was slightly better, about 11.53. Basic congress and outlier indexing have similar speedups as uniform sampling. We note that for all these methods, even though the sample tables are smaller than the original fact table by a factor of 100, it is difficult to achieve a 100x speedup since most queries involve multi-table joins of fact and dimension tables and the dimension tables have not been reduced by the same factor.

Figure 9 shows the average speedup of small group sampling as a function of the number of grouping columns referenced in the query. As can be seen, the speedup decreases as the number of grouping columns increase. This is because a query with more grouping columns requires more small group sample tables during execution. Nevertheless, even for queries with 4 grouping columns, the average speedup is reasonable, about 8x.

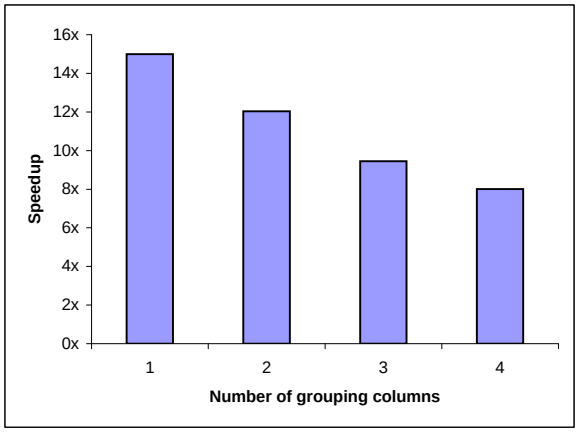


Figure 9: Speedup of Small Group on TPCH5G1.5z

5.4.2 *Preprocessing Performance*

In this paper we started with the assumption that relatively large preprocessing times and space usage are generally acceptable as long as they are not exorbitant. We demonstrate here that the preprocessing requirements of small group sampling are indeed not exorbitant.

We compared both the space and time taken to build the sample tables for all the algorithms. The space consumed by the sample tables for small group sampling is more than the other algorithms as it has to build multiple sample tables. Nevertheless, the space overhead is still quite reasonable: at a base sampling rate of 1%, the total overhead was about 6% of the original database size for the `TPCHxGyz` databases and about 18% for the `SALES` database. In practice, this amount can be further reduced in several ways, such as (a) by using smaller base sampling rates (e.g. reducing the base sampling rate to 0.25% reduced the space overhead to about 1.8% for the `TPCHxGyz` databases), and (b) by considering fewer columns (e.g. available workloads may be analyzed to eliminate infrequently referenced grouping columns).

The preprocessing times for uniform random sampling and

outlier indexing are very short; the respective sample tables are built within a few minutes even for the TPCH5Gyz databases. Basic congress is substantially slower but not exorbitant, with our implementation taking about 3 hours each for the SALES and TPCH1Gyz databases, and 12 hours for the TPCH5Gyz databases. We used a straightforward, unoptimized implementation of small group sampling instead of the more efficient two-pass implementation discussed in Section 4.2.1. Its performance was in the same ballpark as basic congress: about 2.5 hours for the SALES and TPCH1Gyz databases, and about 13.6 hours for the TPCH5Gyz databases. As future work we plan to experiment with the more efficient implementation of small group sampling.

6. CONCLUSION

In this paper we introduced the dynamic sample selection architecture for producing fast, approximate answers to analysis queries over massive databases. Dynamic sample selection improves on previous AQP approaches by its ability to productively utilize additional disk space, which is frequently available in relative abundance, without increasing query response time. We also described small group sampling, a type of dynamic sample selection that targets aggregation queries with group-bys, and gave experimental results showing that small group sampling outperforms alternative techniques.

Not every AQP application is appropriate for dynamic sample selection; for example, when summary structure must be kept small, e.g. for transmission over a wide-area network, then creating a single biased sample or a similar small synopsis is likely to be the best approach. However, we believe that for a large and important class of applications involving ad hoc queries, query response time is the resource that becomes a bottleneck. For such applications, dynamic sample selection provides a way to improve system performance at little additional cost by taking better advantage of existing system resources.

7. REFERENCES

[1] S. Acharya, P. B. Gibbons, and V. Poosala. Aqua: A fast decision support system using approximate query answers. In *Proc. 1999 Intl. Conf. on Very Large Data Bases*, pages 754–755, Sept. 1999.

[2] S. Acharya, P. B. Gibbons, and V. Poosala. Congressional samples for approximate answering of group-by queries. In *Proc. 2000 ACM SIGMOD*, pages 487–498, May 2000.

[3] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. Join synopses for approximate query answering. In *Proc. 1999 ACM SIGMOD*, pages 275–286, June 1999.

[4] S. Agrawal, S. Chaudhuri, and V. Narasayya. Automated selection of materialized views and indexes for sql databases. In *Proc. 2000 Intl. Conf. on Very Large Data Bases*, pages 496–505, Sept. 2000.

[5] A. Agresti and B. A. Coull. Approximate is better than "exact" for interval estimation of binomial proportions. *The American Statistician*, 52:119–126, 1998.

[6] D. Barbara, W. DuMouchel, C. Faloutsos, P. J. Haas, J. M. Hellerstein, Y. E. Ioannidis, H. V. Jagadish, T. Johnson, R. T. Ng, V. Poosala, K. A. Ross, and K. C. Sevcik. The New Jersey data reduction report. *Data Engineering Bulletin*, 20(4):3–45, 1997.

[7] L. D. Brown, T. Cai, and A. DasGupta. Interval estimation for a binomial proportion. *Statistical Science*, 16:101–133, 2001.

[8] K. Chakrabarti, M. N. Garofalakis, R. Rastogi, and K. Shim. Approximate query processing using wavelets. In *Proc. 2000 Intl. Conf. on Very Large Data Bases*, pages 111–122, Sept. 2000.

[9] S. Chaudhuri, G. Das, M. Datar, R. Motwani, and V. Narasayya. Overcoming limitations of sampling for aggregation queries. In *Proc. 2001 Intl. Conf. on Data Engineering*, 2001.

[10] S. Chaudhuri, G. Das, and V. Narasayya. A robust, optimization-based approach for approximate answering of aggregate queries. In *Proc. 2001 ACM SIGMOD*, pages 295–306, May 2001.

[11] S. Chaudhuri, A. Gupta, and V. R. Narasayya. Compressing sql workloads. In *Proc. 2002 ACM SIGMOD*, pages 488–499, June 2002.

[12] S. Chaudhuri, R. Motwani, and V. Narasayya. On random sampling over joins. In *Proc. 1999 ACM SIGMOD*, pages 263–274, June 1999.

[13] S. Chaudhuri and V. Narasayya. Program for tpc-d data generation with skew. Available at ftp://ftp.research.microsoft.com/pub/users/viveknar/tpcdskew.

[14] S. Chaudhuri and V. Narasayya. Efficient cost-driven index selection tool for microsoft sql server. In *Proc. 1997 Intl. Conf. on Very Large Data Bases*, pages 146–155, Aug. 1997.

[15] V. Ganti, M. Lee, and R. Ramakrishnan. Icicles: Self-tuning samples for approximate query answering. In *Proc. 2000 Intl. Conf. on Very Large Data Bases*, pages 176–187, Sept. 2000.

[16] P. Gibbons, Y. Matias, and V. Poosala. Fast incremental maintenance of approximate histograms. In *Proc. 1997 Intl. Conf. on Very Large Data Bases*, pages 466–475, Aug. 1997.

[17] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *J. Data Mining and Knowledge Discovery*, 1(1):29–53, 1997.

[18] H. Gupta. Selection of views to materialize in a data warehouse. In *Proc. 1997 Intl. Conf. on Database Theory*, pages 98–112, Jan. 1997.

[19] H. Gupta, V. Harinarayan, A. Rajaraman, and J. D. Ullman. Index selection for OLAP. In *Proc. 1997 Intl. Conf. on Data Engineering*, pages 208–219, Apr. 1997.

[20] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *Proc. 1996 ACM SIGMOD*, pages 205–216, June 1996.

[21] J. Hellerstein, M. Franklin, S. Chandrasekaran, A. Deshpande, K. Hildrum, S. Madden, V. Raman, and M. A. Shah. Adaptive query processing: Technology in evolution. *IEEE Data Engineering Bulletin*, 23(2):7–18, June 2000.

[22] J. Hellerstein, P. Haas, and H. Wang. Online aggregation. In *Proc. 1997 ACM SIGMOD*, pages 171–182, May 1997.

[23] Y. E. Ioannidis and V. Poosala. Histogram-based approximation of set-valued query-answers. In *Proc. 1999 Intl. Conf. on Very Large Data Bases*, pages 174–185, Sept. 1999.

[24] Z. Ives, D. Florescu, M. Friedman, A. Levy, and D. Weld. An adaptive query execution system for data integration. In *Proc. 1999 ACM SIGMOD*, pages 299–310, June 1999.

[25] Y. Matias, J. Vitter, and M. Wang. Dynamic maintenance of wavelet-based histograms. In *Proc. 2000 Intl. Conf. on Very Large Data Bases*, pages 101–110, Sept. 2000.

[26] V. Raman and J. M. Hellerstein. Partial results for online query processing. In *Proc. 2002 ACM SIGMOD*, pages 275–286, June 2002.

[27] Transaction Processing Performance Council. TPC-H benchmark 1.1.0, June 1999. http://www.tpc.org.

[28] J. Vitter. Random sampling with a reservoir. *ACM Trans. on Mathematical Software*, 11(1):37–57, 1985.