On Sampling and Relational Operators

Surajit Chaudhuri
Microsoft Research
surajitc@microsoft.com

Rajeev Motwani
Stanford University
rajeev@cs.stanford.edu

Abstract

A major bottleneck in implementing sampling as a primitive relational operation is the inefficiency of sampling the output of a query. We highlight the primary difficulties, summarize the results of some recent work in this area, and indicate directions for future work.

1 Introduction

Data warehouses based on relational databases are becoming popular. The investment in data warehouses is targeted towards developing decision support applications that leverage the massive amount of data stored in data warehouses for a variety of business applications. On Line Analytical Processing (OLAP) and data mining are tools for analyzing large databases that are gaining popularity. Many of these tools serve as middleware or application servers that use a SQL database system as the backend data warehouse. They communicate data retrieval requests to the backend database through a relational (SQL) query. On a large database, the cost of executing such ad-hoc queries against the relational backend can be expensive. Fortunately, many data mining applications and statistical analysis techniques can use a sample of the data requested in the SQL query without compromising the results of the analysis. Likewise, OLAP servers that answer queries involving aggregation (e.g., "find total sales for all products in the NorthWest region between 1/1/98 and 1/15/98") can significantly benefit from the ability to present to the user an approximate answer computed from a sample of the result of the query posed to the relational database. It is well-known that for results of aggregation, sampling can be used accurately and efficiently. However, it is important to recognize that whether for data mining, OLAP, or other applications, *sampling must be supported on the result of an arbitrary SQL query*, not just on stored relations. For example, the preceding example of the OLAP query uses a star join between three tables (date, product, and sales).

This paper is concerned with supporting random sampling as a primitive operation in relational databases. In principle, this is easy — introduce into SQL an operation $\textsc{sample}(R, f)$ which produces a uniform random sample S that is an f-fraction of a relation R. While producing a random sample from a relation R is not entirely trivial, it is a well-studied problem and efficient strategies are available [15]. However, these techniques are not effective if sampling needs to be applied to a relation R produced by a query Q rather than to a base relation. It seems grossly inefficient to evaluate Q, computing the entire relation R, only to throw away most of it when applying $\textsc{sample}(R, f)$. It would be much more desirable and efficient to *partially* evaluate Q so as to generate only the sample of R.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

For this purpose, it suffices to consider the case where we are given a query tree T with $\text{SAMPLE}(R, f)$ only at the root. In this setting, it seems plausible that tremendous gains in efficiency can be achieved by "pushing" the sample operation down the tree towards the leaves, since then we would be feeding only a small (random) fraction of the relations (stored as well as intermediate relations) into the query tree and thereby minimizing the cost of query evaluation. To this end, we need to be able to "commute" the sample operation with standard relational operations.

There has not been much past work on supporting sampling as an operation for the end-user of a database system. While random sampling has been proposed and used in many different ways in databases [14, 15], the main focus has been on the use of random sampling for the purposes of estimating query result size, aggregate values, and parameters for query optimization [16, 13, 10, 11, 7, 6, 3].

2 Difficulty of Sampling and Possible Solutions

In this section we elucidate some of the main difficulties in efficiently implementing sampling in a relational system and suggest some avenues for circumventing these difficulties.

2.1 Join

We focus primarily on developing a technique for commuting sampling with a *single* join operation, since we could apply this technique repeatedly to push down the sample operator from the root to the leaves of a join tree. We establish that it is not possible to produce a sample of the result of even a single join from random samples of the two relations participating in the join, and show how we can leverage database statistics to circumvent this impossibility result.

Example 1: Suppose that we have the relations

$$R_1(A,B) = \{(a_1, b_0), (a_2, b_1), (a_2, b_2), (a_2, b_3), \ldots, (a_2, b_k)\},$$

$$R_2(A,C) = \{(a_2, c_0), (a_1, c_1), (a_1, c_2), (a_1, c_3), \ldots, (a_1, c_k)\}.$$

That is, R_1 is defined over the attributes A and B; amongst its $n_1 = k + 1$ tuples, one tuple has the A-value a_1 and k tuples have the A-value a_2, but all have distinct B-values. Similarly, R_2 is defined over the attributes A and C; amongst its $n_2 = k + 1$ tuples, k tuples have the A-value a_1 and one tuple has the A-value a_2, but all have distinct C-values. Observe that their join over A, $J = R_1 \bowtie R_2$, is of size $n = 2k$ and has k tuples with A-value a_1 and k tuples with A-value a_2.

Assume that we wish to choose a random sample. Consider a random sample $S \subseteq J$. We expect that roughly half of the tuples in S have A-value a_1, and roughly half of the tuples in S have A-value a_2.

Suppose we pick random samples $S_1 \subseteq R_1$ and $S_2 \subseteq R_2$. It is quite unlikely that S_1 will contain the tuple (a_1, b_0), or that S_2 will contain the tuple (a_2, c_0). Thus, given the samples S_1 and S_2, it is impossible to generate a random sample of $J = R_1 \bowtie R_2$ for any reasonable sampling fraction. Note that this conclusion holds even if we allow (say) S_2 to be all of R_2 but require that S_1 be a *proper* subset of R_1. In fact, in all these cases we would expect $S_1 \bowtie S_2$ to be empty.

This suggests that SAMPLE does not commute with join since $\text{SAMPLE}(R_1, f_1) \bowtie \text{SAMPLE}(R_2, f_2)$ may not even contain any non-trivial size subset of J, and so further computation or sampling from it cannot be used to extract a sample of J. We formalize this intuition as follows. Suppose we are given samples $S_1 = \text{SAMPLE}(R_1, f_1)$ and $S_2 = \text{SAMPLE}(R_2, f_2)$. Given only S_1, S_2, and any desired set of statistics for R_1 and R_2, but without direct access to the tuples in R_1 and R_2, we are required to produce a sample $S = \text{SAMPLE}(R_1 \bowtie R_2, f)$ for some value of $f > 0$. The theorem below states the extremely negative result implicit in Example 1 — even when we are given *arbitrarily large samples* from R_1 and R_2, as well as *arbitrarily detailed statistics*, it is still not possible to generate any *non-empty* random sample of $R_1 \bowtie R_2$

Theorem 1: Suppose that at least one of f_1, f_2 is strictly less than 1. Then, it is not possible to generate the sample $S = \text{SAMPLE}(R_1 \bowtie R_2, f)$ from S_1 and S_2 for any $f > 0$ where $S_1 = \text{SAMPLE}(R_1, f_1)$ and $S_2 = \text{SAMPLE}(R_2, f_2)$.

Despite this strong negative result, it is still possible to exploit the power of sampling. Our key observation is that given some partial statistics (e.g., histograms) on one of the operand relations, we can use the statistics to *bias* the sampling from the second relation in such a way that it becomes possible to produce a sample of the join. We devise a variety of sampling schemes based on these observations, improving the state-of-the-art for join sampling. In the context of a join tree, our work shows that it is possible to push down the sampling operation to *one* of the two operand relations. At the same time, our negative results above show that it is inherently difficult to achieve greater efficiency by pushing sampling down to *both* operands of a join in a query tree.

For the rest of the discussion, we will need to use some notation. Assume that the two relations R_1 and R_2 are of size n_1 and n_2, respectively, and that we are interested in an equi-join with respect to an attribute A. We denote the domain of the attribute A by D. For each value $v \in D$, let $m_1(v)$ and $m_2(v)$ be the number of distinct tuples in R_1 and R_2, respectively, that contain value v in attribute A.

Observe that the impossibility of commuting SAMPLE with join does not preclude the possibility of somehow obtaining $\text{SAMPLE}(R_1 \bowtie R_2, f)$ from *non-uniform samples* of R_1 and R_2. To better understand this point, consider the tuple $t = (a_1, b_0) \in R_1$ and its influence on $R_1 \bowtie R_2$. While $m_1(a_1) = 1$, the set $J_t(R_2)$ has size $m_2(a_1) = k$. Thus, even though a random sample of R_1 is unlikely to pick up the tuple with A-value a_1, half of the tuples in the join J have A-value a_1. This suggests that we sample a tuple $t \in R_1$ of join attribute value v with probability proportional to $m_2(v)$, in the hope that the resulting sample is more likely to reflect the structure of J. This is the basic insight behind most of our strategies in [4]. As an example, consider the Frequency-Partition-Sample strategy [4], based on the insight that the naive sampling strategy performs badly only when the average frequency of attribute values is high enough to make the join size significantly larger than the size of the operand relations. The idea is to partition the operands into two subrelations, one with the high-frequency values and the other with the low-frequency values. For the low-frequency values, we can use the naive sampling strategy, but for the high-frequency values we need to develop more refined approaches as this is precisely the set of values for which computing the full join is expensive.

The preceding discussion suggests that we sample tuples from R_1 based on frequency statistics for R_2. This leads us to a natural classification of the problem and applicability of sampling techniques based on availability of indexes and statistics on operand relations. We remark that the sampling strategy due to Olken et al [14, 15] applies only to the case where indexes and statistics are available on both operands of the join. In contrast, our hybrid sampling technique does not need a full index on R_2 but merely some partial statistics (on the high frequency values).

The preceding discussion was primarily concerned with the issue of sampling from a single join operation. We now turn to the question of implementing sampling as a primitive relational operation in the context of a join tree. Suppose we have a query Q generating a join tree T with a sampling operation applied to its result R. The earlier negative results state that it is essentially impossible to push down the sampling operation to both operands of a join operation, shedding some light on the difficulty of dealing with linear and arbitrary join trees. Consider sampling from $Q = (R_1 \bowtie R_2 \bowtie R_3)$ - we cannot just select a uniform random sample of $R_1 \bowtie R_2$ but have to pick a non-uniform sample whose distribution depends on the statistics of R_3. But then, what about pushing the sampling further down the tree to R_1? Now, we will have to sample from R_1 using statistics for both R_2 and R_3. In principle, this can be done (e.g., using multidimensional histograms on $\{R_1, R_2\}$), since the operand relations are all base relations and their statistics can be precomputed. However, such requirements provide a high bar on being able to exploit sampling "on-the-fly", i.e., on ad-hoc queries.

Finally, consider the important special case where the query consists of *foreign key* joins, e.g., $R_1 \bowtie R_2$ where R_2 is the dimension (referenced) table. In such cases, each tuple from R_1 joins with at most one tuple in R_2. This allows us to uniformly sample R_1 to create a sample for $R_1 \bowtie R_2$. Unfortunately, in most cases, such queries would contain selection conditions on R_2. As the next subsection shows, presence of selections limits our ability to use sampling.

2.2 Select and Group By

Since most queries involve selection conditions, it is important to study the interaction of sampling with selection. The main insight here is that if the selectivity of a query is low, then it dramatically and adversely impacts the accuracy of sampling-based estimation. Consider a simple query *select count(*) from lineitem*. Suppose we wish to use sampling to approximate the query result. Consider a sample where each tuple is included with probability p (the sampling fraction). If the number n of tuples in the table is large, then the expected number of tuples in the sample (i.e., np) is large and we know from Chernoff bounds [12] that the actual number of tuples in the sample is very close to the expected value with very high probability. As a result, we can accurately estimate the size of the table from the sample.

Suppose, however, we were faced with the following query: *select count(*) from lineitem where city = "Palo Alto"*. Let n_s be the number of tuples that satisfy the selection condition in the query. If n_s is not large enough, then the expected value of the number of tuples from the sample that satisfy the selection condition (i.e., $n_s p$) is low and the variance is large compared to this expected value. In that case, we will incur a large error in query estimation based on the number of sampled tuples passing the selection condition. This intuition is formalized in the following theorem due to Chaudhuri, Datar, Motwani, and Narasayya [5] where we show that the relative error for a query is proportional to the inverse square root of the selectivity of the conditions in the query. Therefore, we need a significantly larger sample for queries with low selectivity. Earlier work has not satisfactorily addressed this issue. Acharya et al [1] maintain samples of varying size and use different samples for different queries. This requires estimating the selectivity of a query accurately. If we underestimate the selectivity, we will end up using a bigger sample than required, thereby decreasing efficiency. On the other hand, if we overestimate the selectivity, we will end up using a smaller sample and incurring a loss in accuracy. Likewise, the technique adopted by Hellerstein et al [9] will fail to obtain good confidence bounds unless we have looked at a large part of the relation. Thus, low selectivity is a serious problem that limits the use of sampling as a robust technique.

Although we have focussed exclusively on Selection, a careful reader will note that the effect of *Group By* is no different from that of selection since the effect of Group By is to create multiple selection queries that differ by virtue of their qualifying values on the grouping attribute. Therefore, similar issues as has been studied for Selection arises.

2.3 Select Distinct

We now turn to the problem of estimating the number of distinct values in a column of a table. Our objective is to provide estimation for such aggregates without scanning the entire table. A natural idea is to use a random sample of the data. Unfortunately, we [3] proved that large errors are unavoidable for estimates derived only from uniform random samples unless the sample size is very close to the size of the table itself. Indeed, we have generalized this result and proved a stronger negative result for the the most general possible class of estimators that, instead of being restricted to merely a random sampling, are allowed to use *any* (possibly randomized) strategy to select a sequence of r values to examine in the the input table. Such estimators could even pick a sequence of r tuples *adaptively*, i.e., pick the next tuple to examine based on the values of the previously-examined tuples. This explains the observation by Haas et al [8] that no known estimator behaves well on all data distributions primarily because their performance is sensitive to the skew of the data. Indeed, our lower bound on the error stems in part from the difficulty in distinguishing between low-skew and high-skew data. In fact, the proof hinges on the inability to distinguish between two specific scenarios: one where there are few distinct values of very high frequency, and another where there a few high-frequency values together with a large number of very low-frequency values (thus having a large number of distinct values). The first scenario has low skew with few distinct values and the second one has high skew with a large number of distinct values. The following theorem, due to Charikar, Chaudhuri, Motwani, and Narasayya [2], states that any possible estimator must incur large error on at least one of these two scenarios.

Theorem 2: Consider any (possibly adaptive and randomized) estimator $\widehat{D}$ for the number of distinct values D that examines at most r rows in a table with n rows. Then, for any $\gamma > e^{-r}$, there exists a choice of the input data such that with probability at least γ,

$$\text{error}(\widehat{D}) \geq \sqrt{\frac{n-r}{2r} \ln \frac{1}{\gamma}}.$$

Our error bounds seem to be close to the errors observed in real experiments, even though our results give worst-case bounds and the data in the experiments may have had special structure that could have been used to obtain better estimates. However, in [2], we were also able to provide a robust estimator that has an error bound that matches the lower bound above.

2.4 Aggregation

So far we have restricted our focus to queries that count the number of tuples, e.g., *select count(*) From sales_table*. However, in many cases, we are interested in other aggregating functions in addition to *count*, e.g., *select sum(sales) from sales_table*. For such aggregates, any skew present in the data may have a severe impact on use of sampling. The following example is instructive.

Example 2: Suppose we have $10,000$ tuples of which 99% have value 1 in the aggregate column, while the remaining 1% of the tuples have value 1000. Consider using a uniform random sample of size 100 to estimate the average value of the column over all tuples. It is quite likely that the sample would not include any tuple of value 1000, leading to an estimate of 1 for the average value. On the other hand, if perchance two or more tuples of value 1000 were to be included in the sample, then our estimate of the average value would be more than 20.98. In either case, the estimate would be far from the true value of the average which is 10.99. It is only in the case where we get exactly one tuple of value 1000 in the sample that we would obtain a reasonable estimate of the average value. But the latter event has probability 0.37, therefore with high probability we would get a large error in the estimate.

As in the above example, a skewed database is characterized by the existence of certain tuples that are deviant from the rest with respect to their aggregate value. We refer to these tuples as *outliers*. We observe that we cannot afford to miss these outliers in the sample. It should be noted that the presence of these tuples in the wrong proportion in the sample would also lead to large errors in the estimate of a query's result. In Datar, Motwani, and Narasayya [5], we show that the relative error is proportional to the ratio C_{MAX}/C_{AVG}, where C_{MAX} and C_{AVG} are the maximum and average values in the data set. This ratio increases with the skew in the data. Not surprisingly, the approach to handle such skew is to identify tuples that are outliers [5]. The earlier work of Acharya et al [1] does not address this issue. In the work of Hellerstein et al [9], they assume that the aggregate attributes are not skewed and the experiments reported in their paper are over grades data where there is not much variation. The confidence intervals provided with their estimate could be severely affected by the presence of skew.

3 Practicality of Sampling

Random sampling is an attractive data-reduction operation that reduces the cost of executing complex ad-hoc queries on large databases. But the preceding discussion shows that there are non-trivial barriers to implementing sampling as a relational operator. While the techniques we reviewed are able to circumvent some of these barriers, the goal of pushing sampling down to the leaf nodes of a complex query tree requires a substantial increase in the sample size to keep the error within bounds. In effect, sampling the results of an ad-hoc query is an inherently difficult and unsolved problem. Our work has demonstrated that auxiliary information such as histograms can be leveraged to facilitate efficient sampling. It remains an interesting question to determine what other auxiliary information may be employed for this purpose and more broadly for computing approximate answers to relational queries.

References

[1] S. Acharya, P. Gibbons, V. Poosala, and S. Ramaswamy. Join Synopses for Approximate Query Answering. In *Proc. ACM SIGMOD Conference,* 1999.

[2] M. Charikar, S. Chaudhuri, R. Motwani, and V. Narasayya. Towards Estimation Error Guarantees for Distinct Values. Submitted for publication, 1999.

[3] S. Chaudhuri, R. Motwani, and V. Narasayya. Using Random Sampling for Histogram Construction. In *Proc. ACM SIGMOD Conference,* pages 436–447, 1998.

[4] S. Chaudhuri, R. Motwani, and V. Narasayya. On Random Sampling Over Joins. In *Proc. ACM SIGMOD Conference,* pages 263–274, 1999.

[5] S. Chaudhuri, M. Datar, R. Motwani, and V. Narasayya. Overcoming Limitations of Sampling for Aggregation Queries. Submitted for publication, 1999.

[6] S. Ganguly, P.B. Gibbons, Y. Matias, and A. Silberschatz. Bifocal Sampling for Skew-Resistant Join Size Estimation. In *Proc. ACM SIGMOD Conference,* pages 271–281, 1996.

[7] P.J. Haas, J.F. Naughton, and A.N. Swami. On the Relative Cost of Sampling for Join Selectivity Estimation. In *Proc. 13th ACM PODS,* pages 14–24, 1994.

[8] P.J. Haas, J.F. Naughton, S. Seshadri, and L. Stokes. Sampling-based estimation of the number of distinct values of an attribute. In *Proc. 21st VLDB,* pages 311–322, 1995.

[9] J.M. Hellerstein, P.J. Haas, and H.J. Wang. Online Aggregation. In *Proc. ACM SIGMOD Conference,* pages 171–182, 1997.

[10] W. Hou, G. Ozsoyoglu, and E. Dogdu. Error-Constrained COUNT Query Evaluation in Relational Databases. In *Proc. ACM SIGMOD Conference,* pages 278–287, 1991.

[11] R.J. Lipton, J.F. Naughton, D.A. Schneider, and S. Seshadri. Efficient Sampling Strategies for Relational Database Operations. *Theoretical Computer Science* 116(1993): 195–226.

[12] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[13] J.F. Naughton and S. Seshadri. On Estimating the Size of Projections. In *Proc. Third International Conference on Database Theory,* pages 499–513, 1990.

[14] F. Olken and D. Rotem. Simple random sampling from relational databases. In *Proc. 12th VLDB,* pages 160–169, 1986.

[15] F. Olken. *Random Sampling from Databases.* PhD Dissertation, Computer Science, University of California at Berkeley, 1993.

[16] G. Piatetsky-Shapiro and C. Connell. Accurate estimation of the number of tuples satisfying a condition. In *Proc. ACM SIGMOD Conference,* pages 256–276, 1984.