OVERCOMING LIMITATIONS OF SAMPLING FOR AGGREGATION QUERIES

Surajit Chaudhuri	**Microsoft Research**
Gautam Das	**Microsoft Research**
Mayur Datar	**Stanford University**
Rajeev Motwani	**Stanford University**
Vivek Narasayya	**Microsoft Research**

Presented by :-

Vinit Asher	***1000570792***
Deep Pancholi	***1000556121***

INTRODUCTION

- Why sampling?
- Why uniform random sampling?
- Uniform Random Samples and factors responsible for significant error introduction
  - Skewed database i.e. characterized by presence of Outlier values
  - Low selectivity of queries (related to aggregate queries).
- How do we overcome the problems?

TECHNIQUES SUGGESTED IN PAPER

- For problems due to skew in data, the paper recommends to isolate the values in the dataset that contribute heavily to error in sampling.

- Exploit the workload information to overcome limitations in answering queries with low selectivity.

KEY CONTRIBUTIONS OF THE PAPER

- Experimental evaluation of the proposed techniques based on implementation on Microsoft SQL Server

- Paper also demonstrates that a combination of outlier indexing and weighted sampling results in significant error reduction compared to Uniform Random Sampling.

EXAMPLE OF DATA SKEW AND ITS ADVERSE EFFECT

- Suppose N=10,000 such that 99% tuples have a value of 1, remaining 1% have a value of 1000.
- Hence, Sum of this table is 9900+100,000=109,900
- Suppose we take a Uniform Random Sample of size 1%. Hence, n=100
- Now it is possible that all 100 tuples in sample have a value 1 giving us a sum of 100*100=10,000 which is way way less than correct answer.
- If we get 99 tuples with value 1 and 1 tuple with 1000, sum is 1099*100=109,900. Similarly for 98 tuples of 1 and 2 tuples of 1000, sum=209,800
- This shows that if we get a value of 1000 in sample more than once, we are going to get a huge error. And not getting any 1000 value also causes a huge error.
- However, the probability to get just 1 value of 1000 in sample is mere 0.37!!! It means the probability to get an erroneous result is 0.63.

EFFECT OF DATA SKEW (CONTINUED)

- Similar arguments also hold for the aggregate average.
- Tuples deviant from the rest of the values in terms of their contributions to aggregrate are known as outliers.

EFFECT OF DATA SKEW (CONTINUED)

- **Theorem 1**: For a relation R with elements $\{y_1, y_2,\ldots,y_N\}$, let U be the uniform random sample of size n. Then actual sum will be $Y = \sum_{i=1}^{N} y_i$ and the unbiased estimate of the actual sum will be $Y_e=(N/n)\sum_{yi\in U} y_i$ with a standard error as follows:

$$\epsilon = \frac{NS}{\sqrt{n}}\sqrt{1-\frac{n}{N}}$$

Where S is the standard deviation of the values in the relation defined as $S = \sqrt{\frac{\sum_{i=1}^{N}(y_i - \overline{Y})^2}{N-1}}$

- This shows that if there are outliers in the data, then S could be very large. In this case, for a given error bound, we will need to increase the sample size n.

EFFECT OF LOW SELECTIVITY AND SMALL GROUPS

- What happens if the selectivity of a query is low?
  - It adversely impacts the accuracy of the sampling based estimation
- A selection query partitions the relation into two sub-relations as follows:
  - Tuples that satisfy the condition of select query
  - Tuples that don't satisfy the condition.
- When we sample uniformly, the number of tuples that are sampled from the relevant sub-relation are proportional to its size.
- Hence, if the relevant sample size is low, it can lead to huge errors.
- For Uniform random sampling to perform well, the relevant sub-relation should be large in size, which is not the case in general.

HANDLING DATA SKEW: OUTLIER INDEXES

- Outliers/deviants in data ⇨ Large Variance ⇨ High errors
- Hence we identify tuples with outlier values and store them in separate sub relation.
- The proposed technique would result in a very accurate estimation of the aggregate.
- In Outlier indexing method, a given relation R is partitioned as R_O (Outliers) and R_{NO} (no-outliers). The Query Q now can be run as a union of two sub-queries one on R_O and another on R_{NO}.

HANDLING DATA SKEW: OUTLIER INDEXES (EXAMPLE)

- **Preprocessing steps**
  - Determine Outliers – *Specify sub-relation R_0 of R to be the set of outliers*
  - Sample Non-Outliers – *Select a uniform random sample T of the relation R_{NO}*
- **Query processing steps**
  - Aggregate outliers – *Apply the query to outliers in R_0*
  - Aggregate non-outliers – *Apply the query to sample T and extrapolate to obtain an estimate of the query result for R_{NO}*
  - Combine Aggregates – *Combine the appoximate result for R_{NO} with the exact result for R_0 to obtain an appoximate result for R*

SELECTION OF OUTLIERS

- In this method of Outlier-indexing, query error is solely because of error in estimating non-outlier aggregation.
- However, there is additional overhead for maintaining and accessing an outlier index.
- Theorem 2: Consider a multiset R = $\{y_1, y_2, \ldots, y_N\}$ in sorted order. Let $R_O \subset R$ be the subset such that • $|R_O| \leq \tau$, and • $S(R \setminus R_O) = min_{R' \subset R, |R'| \leq \tau} \{S(R \setminus R')\}$ *Then, there exists some* $0 \leq \tau' \leq \tau$ *such that* $R_O = \{y_i | 1 \leq i \leq \tau'\} \cup \{y_i | (N + \tau' + 1 - \tau) \leq i \leq N\}$
- Here τ is the allocated memory for outlier indexing.

ALGORITHM FOR OUTLIER INDEX (R,C, τ)

- Alogirithm Outlier-Index(R,C, τ):
- Let the values in column C be sorted in relation R
- For i = 1 to τ+1, compute

  E(i) = S({yi, yi+1,…yN- τ+i-1})
- Let i' be the value of i where E(i) is minimum.
- Outlier-index is the tuples that correspond to the set of values $\{yj \mid 1 \leqslant j \leqslant \tau'\} \cup \{yj \mid (N+\tau'+1-\tau) \leqslant j \leqslant N\}$ i.e. lowest 1 to τ' values and highest $\tau - \tau'$ values.
- where $\tau' = i' - 1$
- It is possible to give standard error (probabilistic) guarantee of our estimated answer using Theorem1.

STORAGE ALLOCATION FOR OUTLIER INDEXING

- Given sufficient space to store m tuples, how do we allocate storage between samples and outlier-index so as to get minimum error?
- Suppose S(t) denotes standard deviation in non-outliers for optimal outlier index of size t.
- From Theorem 1, error is proportional to $S(\tau)/\sqrt{m-\tau}$ where t tuples are in outlier index and m-t tuples in the sample.

EXTENSION OF OUTLIER INDEXING TO OTHER AGGREGATES

- For *count* aggregate the outlier indexing is not beneficial since there is no variance among the data values.
- In case of aggregate *avg (average)*, during query processing, an *avg* query is estimated as sum/count
- Outlier-indexing is also not beneficial for the aggregates that depend upon the *rank* of tuples rather that their actual values (such as *min*, *max* or *median*)

FINAL NOTES ON OUTLIER INDEXING

- Outlier indexing technique works best for aggregations on a single table that don't involve foreign-key joins.
- It is beneficial for *sum* and *average* aggregations but doesn't work for rank-order based queries such as *max, min or count*.
- As future work, investigation is going on whether to create separate outlier indexes for frequently occurring functions depending on the workload information.

HANDLING LOW SELECTIVITY AND SMALL GROUPS

- In this case, we want to use weighted sampling. In other words, we want to sample more from subsets of data that are small in size but are important (i.e. having high usage).
- We select a representative workload (i.e. a set of queries) and tune the sample so that we can answer the queries posed to database more accurately.
- The technique mentioned in the paper is for precomputed samples only. However, research is being done to apply this solution to online sampling.

EXPLOITING WORKLOAD INFORMATION

- *Workload Collection:* It means obtaining a workload consisting of representative queries that are posed to the database. Tools like profiler component in the Microsoft SQL Server allow logging of queries posed to the Database.
- *Tracing Query Patterns:* It basically involves analyzing the workload to obtain parsed information.
- *Tracing Tuple Usage:* It involves tracing number of times a tuple was accessed, number of times it satisfies the query, number of times tuple didn't satisfy the condition, etc.
- *Weighted Sampling:* It involves sampling by taking into account weights of tuples into consideration.

WEIGHTED SAMPLING: DEEPER INSIGHT

- Same as stratified sampling done in class. All the tuple are assigned some weightage depending on traces tuple usage
- Whenever we want to get a sample, a tuple is associated in the sample with probability $p_i = n * w_i'$
- Now, as discussed in class, the inverse of p_i is the multiplication factor. Each aggregate computed over this tuple gets multiplied by the multiplication factor to answer the query. As the uniform random sample has equal probability, the multiplication factor is (N/n).
- This method works well only if we have a workload that is a good representation of the actual queries posed in future and also if access pattern of the queries is local in nature.

IMPLEMENTATION AND EXPERIMENTAL SETUP

- *Databases:* The database used for experimentation is the TPC-R benchmark database. The data generation was modified a bit to get varying degrees of skew. The modified program generates data based on Zipfian distribution.
- *Parameters:*
  - Skew of the data (z) varied over 1, 1.5, 2, 2.5 and 3
  - Sampling fraction (f) varied from 1% to 100%
  - Storage for Outlier index varies 1%, 5%, 10% and 20%.

IMPLEMENTATION AND EXPERIMENTAL SETUP (2)

- The comparisons were done on three options:
  - 1) Uniform Sampling
  - 2) Weighted Sampling
  - 3) Weighted Sampling + Outlier Indexing
- The results depicted here are for which the storage size for outlier-indexing was only 10% of the size of the data set.

EXPERIMENTAL RESULTS

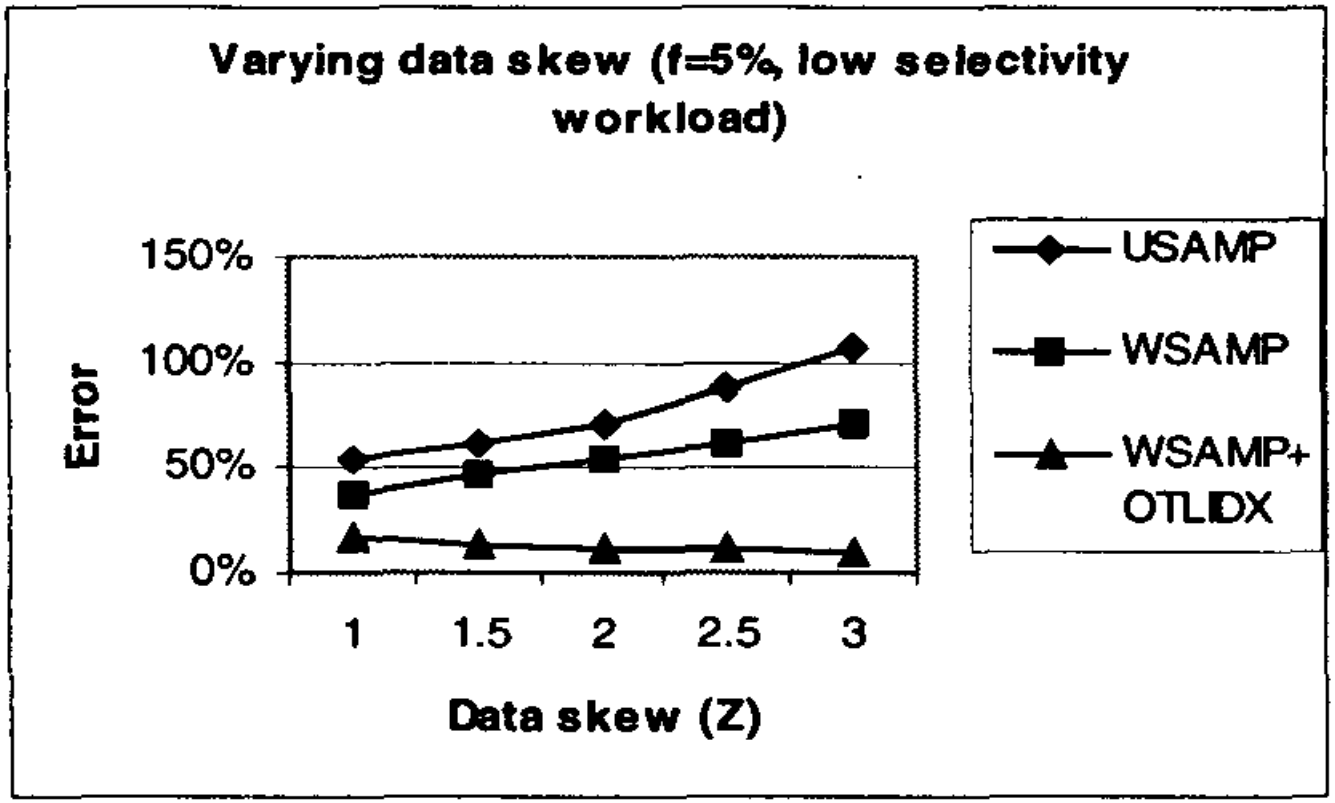

Figure 1. Error versus data skew

EXPERIMENTAL RESULTS (2)

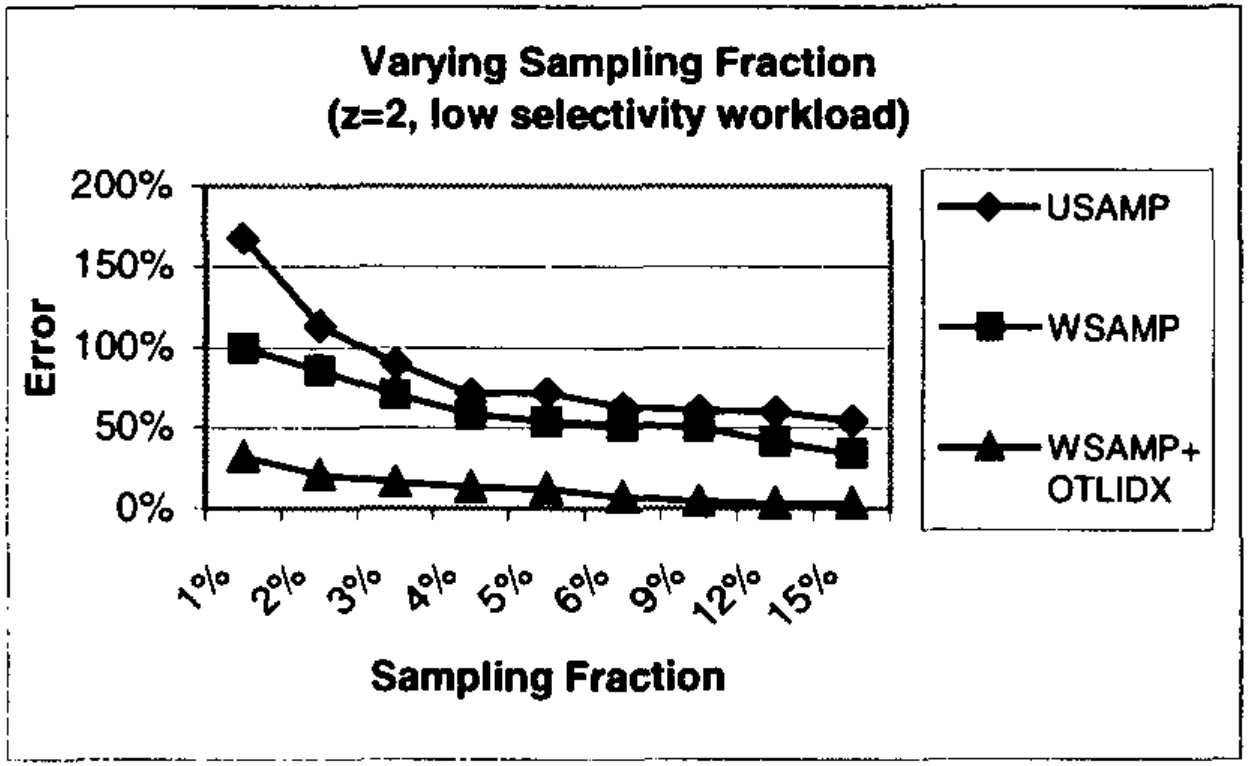

Figure 2. Error versus sampling fraction

EXPERIMENTAL RESULTS (3)

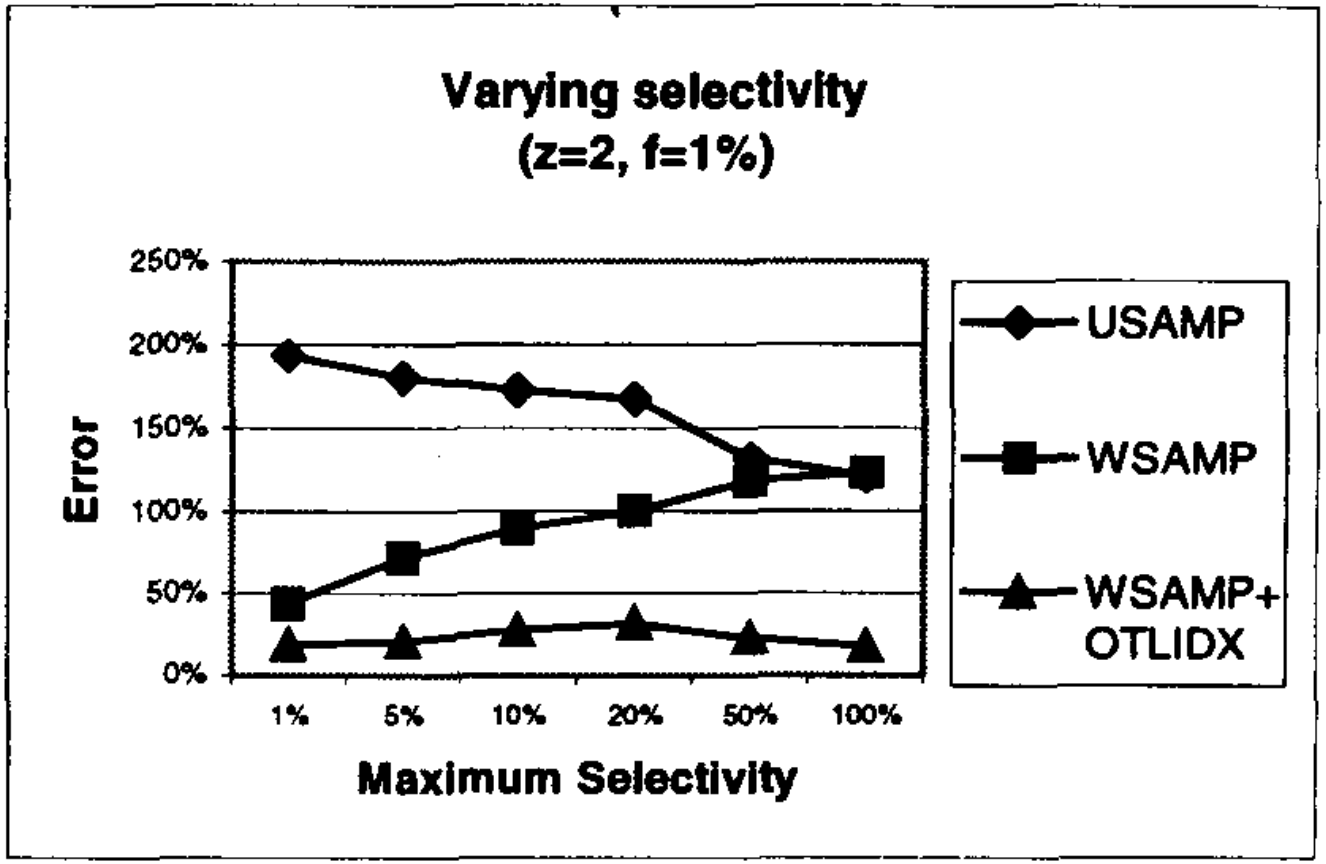

Figure 3. Error versus selectivity of queries

CONCLUSION

- We can easily conclude that skew in the data can lead to considerable large errors. However, Outlier indexing addresses this problem at a small additional overhead. The problem lies in creating a single outlier index that works for any query on the database.
- Low selectivity of queries is addressed by the weighted load of sampling.
- Some important references:
  - http://www.cs.sfu.ca/~wangk/standards.pdf - for more info on TPC-R standard database.

DIRECTION OF FUTURE WORK

- Building of a single outlier-index for different aggregates and aggregate expressions
- Tuning the selection of outlier-index using workload information
- Extension of the techniques discussed to a wider class of join queries