Sampling Bitmap Indices to Speed Up OLAP Queries

CS784 Project Report

Jayavel Shanmugasundaram

May 14, 1998

Contents

1 Introduction and Motivation 1

2 Approximate Query Processing by Sampling Bit Maps 1

2.1 Sampling Queries without the Group By Clause 2

2.2 Sampling Queries with the Group By Clause 3

3 Implementation Details 3

3.1 Sampling Strategies for Bit Map Indices 3

3.2 Implementation in Shore and Paradise 4

4 Performance Results 5

4.1 Sample Size 5

4.2 Number of Groups in Result 6

4.3 Number of Tuples 9

4.4 Bit Versus Byte Sampling Technique 11

5 Conclusions and Future Work 11

1 Introduction and Motivation

Decision support systems are becoming increasingly important applications of database systems and they impose new requirements on system design. Queries arising in such systems are typically read-only, aggregation queries that could potentially access large amounts of data. Further, fast response time is desirable for such queries.

Bit map indices are index structures that are appropriate for such environments [5]. Whenever a query involves selection conditions on attributes indexed by a bit map, the tuples satisfying the selection condition can be identified simply by anding or oring bit maps and retrieving only the selected tuples. The bit map anding and oring functions are typically very fast and if the bit maps are in main memory, the performance gained using bit maps can be dramatic.

One of the problems with traditional query processing using bit map indices is that if the number of tuples to be retrieved is large, then the query can take a long time to run. This is because the time to access the tuples from the secondary storage device can be substantial if a large number of tuples are to be retrieved. In this project, we are driven by the motivation to efficiently handle such queries.

The key observation that we use is the fact that queries accessing a large number of objects and returning a small (aggregated) output can use sampling techniques to give answers that are accurate with a high probability. Thus, if the user is willing to live with approximate answers, that are close to the actual results with a high probability, then the sampling based technique will be effective. In this report, we study the performance gains obtained by sampling bit maps and study the effects of this on the accuracy of the query results. The significant performance gains for some queries suggest that this approach can be used as an alternative to precomputation [4, 6]. One advantage of this approach over precomputation is that the results are more up to date because the query runs over the current data.

The rest of this report is organized as follows. In Section 2, we outline the idea behind sampling bitmaps and explore various methods of sampling from bitmaps. In Section 3, we describe an implementation in the Paradise database system and in Section 4, we present some performance results. We summarize the report and outline areas for future work in Section 5.

2 Approximate Query Processing by Sampling Bit Maps

In this section, we outline the strategies for processing queries by sampling bit maps. We first consider queries without "groupby"s and then consider groupby queries. In this report, we consider only simple aggregation queries such as sum and count.

2.1 Sampling Queries without the Group By Clause

In the discussion that follows, we assume that the bit map corresponding to the set of tuples to be accessed is given. In particular, we do not go into the details of how the bit map for queries without groupby clauses are constructed. For a detailed exposition of this, see [5].

The bit map corresponding to the selection condition of an aggregation query has 1's in the bits corresponding to the tuples satisfying the selection condition and 0's in the bits corresponding to the tuples not satisfying the selection condition of the query. In traditional query processing, all the tuples corresponding to 1 bits are retrieved from the disk and the aggregation performed on the required attribute of these tuples.

In the sampling technique, we sample the bit map and create a new bit map in which the set of tuples which have their corresponding bits set to 1 is a subset of the set of tuples having their corresponding bits set to 1 in the original bitmap. The number of 1's in the sampled bit map, i.e., the sample size, is the number of tuples that will be accessed from disk. If the sample size is significantly smaller than the number of 1's in the original bit map, then significant savings in I/O can be expected because the number of tuples to be accessed from disk decreases dramatically.

Sampling introduces some approximation in the answer to the query because not all the tuples satisfying the selection condition are accessed. In the rest of this section, we give methods used to provide bounds on the answers based on the distribution of the data and the sample size. Our presentation follows that in [2]. Let $SELECT$ be the set of tuples satisfying the selection condition of the query and let $SAMPLE$ be the set of sampled tuples. It is obvious that $SAMPLE \subseteq SELECT$. Let n be the number of tuples in $SELECT$ and let s be the number of tuples in $SAMPLE$. For a tuple t, let t_i be the attribute on which an aggregation operations has to be performed. Then the actual aggregate value is:

$$\text{Actual Total} = A = \sum_{t \in SELECT} (t_i)$$

and the estimated aggregate value is:

$$\text{Estimated Total} = E = \sum_{t \in SAMPLE} (t_i) \times \frac{n}{s}$$

A (nearly) unbiased estimate of the standard error of the estimated total with respect to the actual total is:

$$\text{Standard Error} = e = \frac{n \times s}{\sqrt{s}} \times \sqrt{\frac{n-s}{n}}$$

where:

$$s = \sqrt{\frac{\sum_{t \in SAMPLE} (t_i - E)^2}{s-1}}$$

If we assume that the estimates for the aggregate value are normally distributed, then the lower and upper confidence limits for the aggregate value are:

$$\text{Lower Confidence Limit} = E - t \times e$$

$$\text{Upper Confidence Limit} = E + t \times e$$

where t is the normal deviate corresponding to the desired confidence probability.

The above equations can be used to calculate the estimated aggregate and the confidence interval with respect to that aggregate and these results can be provided with the results of the query.

2.2 Sampling Queries with the Group By Clause

Traditional query processing for processing group by queries [5] use bit map operations to identify the tuples satisfying each group and then access these tuples to compute the aggregate value for each group. One optimization that could be done is to 'or' all the individual group bit maps and then retrieve all the tuples in the selected groups in sequence. A hash based partitioning algorithm could then be used to separate out the tuples in the individual groups. This approach would reduce the disk overhead because all the satisfying tuples are accessed in sequence instead of being accessed in group order.

Merely sampling from the bit map that contains the tuples in the selected groups is not likely to work well in general. This is because, certain groups may contain far more tuples than other groups. Thus, if we sample from the final bit map, then we would expect to see far fewer tuples from the groups having a few tuples than from groups having a large number of tuples. This would lead to greater error for smaller groups, and in the worst case, this would lead to missed groups (if no tuples in a group are selected at all).

A solution to this problem is to sample tuples from each group *before* the bit maps corresponding to each group are 'or'ed to form the final bit map. In this way, we would see the same number of tuples from each group, irrespective of how many tuples are there in the group and we would also ensure that no groups are missed. The estimates of the aggregate and the confidence interval for these estimates for each group can be calculated using the formulae presented in the previous section.

3 Implementation Details

In this section, we explore various sampling strategies for bit maps. We then describe the implementation of sampling based query processing using bit maps in the Paradise database system on top of the Shore object system.

3.1 Sampling Strategies for Bit Map Indices

We outline three sampling strategies for bit maps and then explore one of them in detail. We plan to study the other two sampling methods in greater depth as part of future work (see Section 5).

The three sampling strategies that we consider are the following. We assume that the number

of 1's in the input bit map is n and that the number of 1's in the output bit map is to be approximately s (i.e., s is the sample size).

1. **Range Sampling:** The number of 1's in the input bit map (n) is first determined. s random numbers, $r_1, ..., r_s$, in the range $1...n$ are then generated. The output bit map is constructed by setting a 1 corresponding to each r_i^{th} 1 in the input bit map, where $1 \leq i \leq s$. This strategy is likely to be effective for any distribution of 1's in bit maps.

2. **Direct Sampling:** In this method, random locations in the input bit map are probed to check for 1's. Whenever a 1 is found, the corresponding bit in the output bit map is set. The process stops when the number of 1's set in the output bit map is s. This method is likely to be effective only when the density of 1's in the bit maps is high.

3. **Binary Search:** Bit maps with different densities of 1's are precomputed and stored. A binary search of these precomputed bit maps is performed by 'and'ing the bit maps with the input bit map in order to find a resultant bit map that has close to s 1's set. This is the sampled bit map. The situations under which this method is likely to perform well need to be further investigated.

We explore on the range sampling approach in detail in this report. In order to generate the random number in the range $1...n$, we use a fast algorithm detailed in [7]. This approach requires $O(s)$ time, where s is the number of samples, and requires approximately s uniform random variates and the computation of about s exponentiation operations. Once the s random numbers are generated, there are two approaches to setting the appropriate 1's in the output bit map.

In the first approach, the input bit map is scanned bit by bit and the appropriate 1's are set in the output bit map. This approach is easy to implement and uses very simple operations. However, scanning a bit map bit by bit may be very expensive especially if the number of bits (number of tuples in the input relation) is very large.

The second approach to range sampling is a byte oriented version of the first approach. In this case, the bit map is scanned byte by byte and a table lookup is used to determine the number of 1's in each byte. The appropriate 1's are again set in the output bit map. This approach processes a bit map a byte at a time and is thus likely to be faster than the bit oriented approach. However, more complex operations like table lookup would reduce the expected performance gain.

3.2 Implementation in Shore and Paradise

We implemented the bit map sampling based query processing approach in the Paradise database system [3] built on top of the Shore object system [1]. Bit maps had been implemented as part of Shore and we added a sampling operation on bit maps that created a new bit map that was a sample of the original bit map. The number of samples required could

Parameter	Range of Values	Default Value
Sample Size per Group	25, 50, 100, 200	100
Number of Groups	1, 25, 50, 100	25
Number of Tuples (x 1000)	250, 500, 750, 1000	500
Sampling Technique	Bit-oriented, Byte-oriented	Byte-oriented

Table 1: Parameters for the Experiments

be specified as a parameter to the sampling operator. The range sampling approach detailed in the previous section was used.

Bit map indices and query processing using these indices had been implemented as an ADT in the Paradise database system. We modified the query processing part to sample the bit maps before accessing the tuples from the disk and to scale up the results and calculate error bounds once the aggregation was done. One of the advantages of our approach, which we discerned during the implementation, is that very little modification needs to be made to existing query processing code in order to incorporate sampling in bit maps. Thus, a substantial part of the code could be reused while still being able to add the required functionality. This suggests that the sampling approach could be easily integrated into existing database systems.

4 Performance Results

In this section, we study the performance of queries using the sampling approach and compare this performance with that of traditional query processing. All queries were run on a database having a single relation with four dimension attributes and one value attribute. All attributes are of type integer. For the purposes of the experiments, we grouped on just one of the attributes. The tuples were generated based on a uniform distribution for all the attributes.

Table 1 shows the setting for the various parameters that were varied for the experiments. For each experiment, exactly one of the parameters were varied while the others were set at the default value.

4.1 Sample Size

In this section, we study the effect of sample size on the performance of the query and the accuracy of the result. The accuracy of the query is studied for the case of uniform distribution of attribute values in the input relation. The results may vary for other distributions.

Figure 1 shows the variation in query processing time with respect to variations in the sampling size. As can be seen from the figure, the query processing time slightly increases

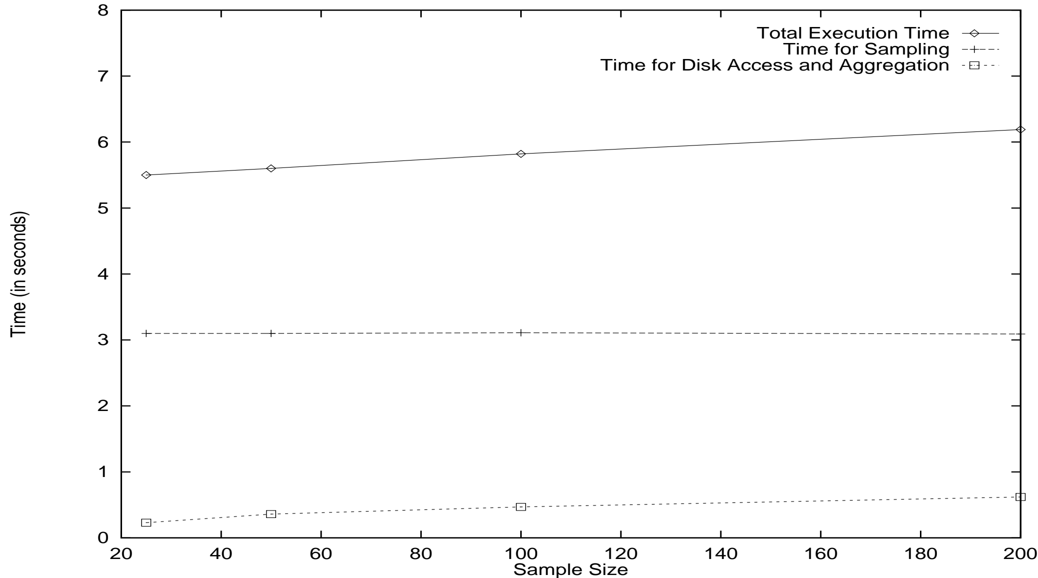

Figure 1: Effect of Sample Size on Performance

with increase in sample size. The main factor that contributes to this increase is the time required for disk accesses and aggregation. This is because more tuples have to be accessed and aggregated. It is interesting to note that the sampling time is relatively independent of the size of the sample because of the efficient sampling algorithm that is used. This suggests that the sample size can be determined based mainly on the required accuracy of the query results.

Figure 2 shows the expected percentage error as a function of the sample size for a 95% confidence interval. As can be seen, the reduction in the error decreases per unit increase in sample size as the sample size increases. Thus increasing the size of the sample after a certain size is unlikely to give significant gains in accuracy.

4.2 Number of Groups in Result

In this section, we study the performance of queries as the number of groups in the result are varied. Figure 3 compares the traditional query processing approach with the sampling based approach. As can be seen, the gain in performance is dramatic when the number of groups in the result are low. This performance gain, however, diminishes as the number of groups in the result increase. The reason for this is explained below.

From Figure 4, it can be seen that the main cost of traditional query processing is in disk accesses (mostly) and the time spent in aggregation. The performance of queries is relatively insensitive to the number of groups because the entire relation is scanned in all cases. The performance of the case where there is only one group is better than the rest because there is no overhead of grouping in this case.

On the other hand, in the case of sampling based query processing (see Figure 5), the

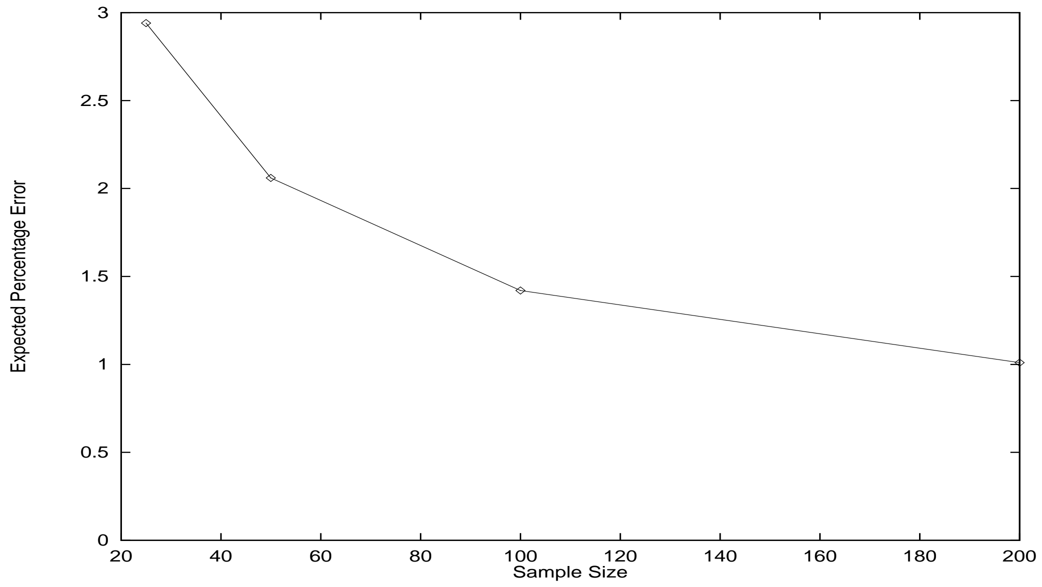

Figure 2: Effect of Sample Size on Expected Error

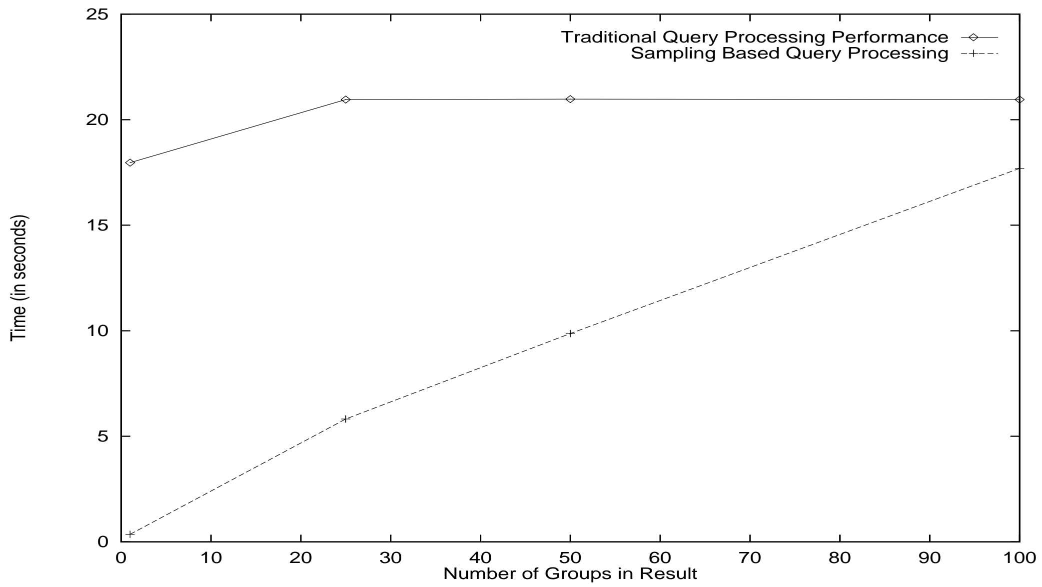

Figure 3: Effect of Number of Groups on Overall Performance

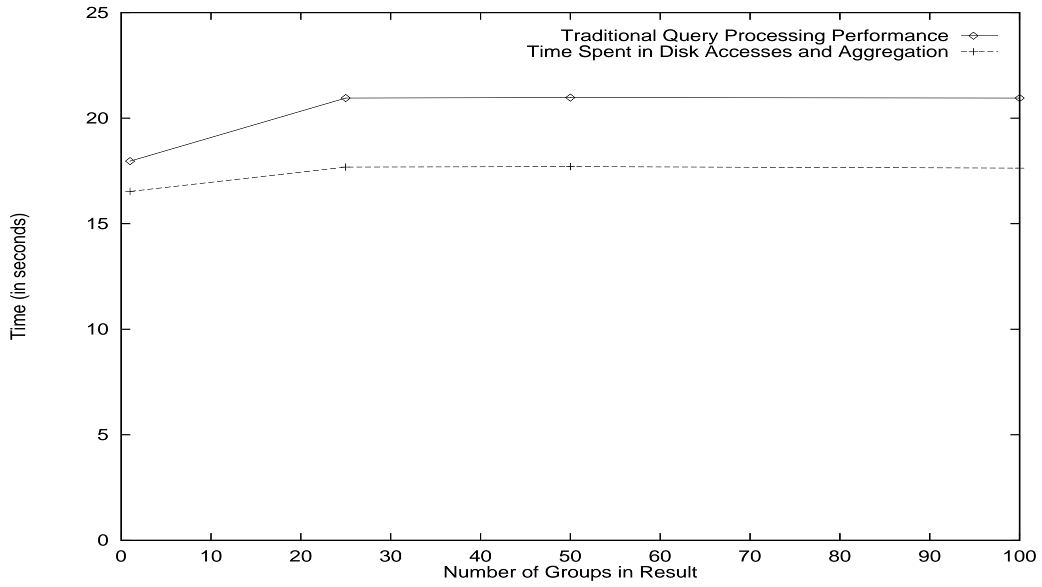

Figure 4: Effect of Number of Groups on Traditional Query Processing

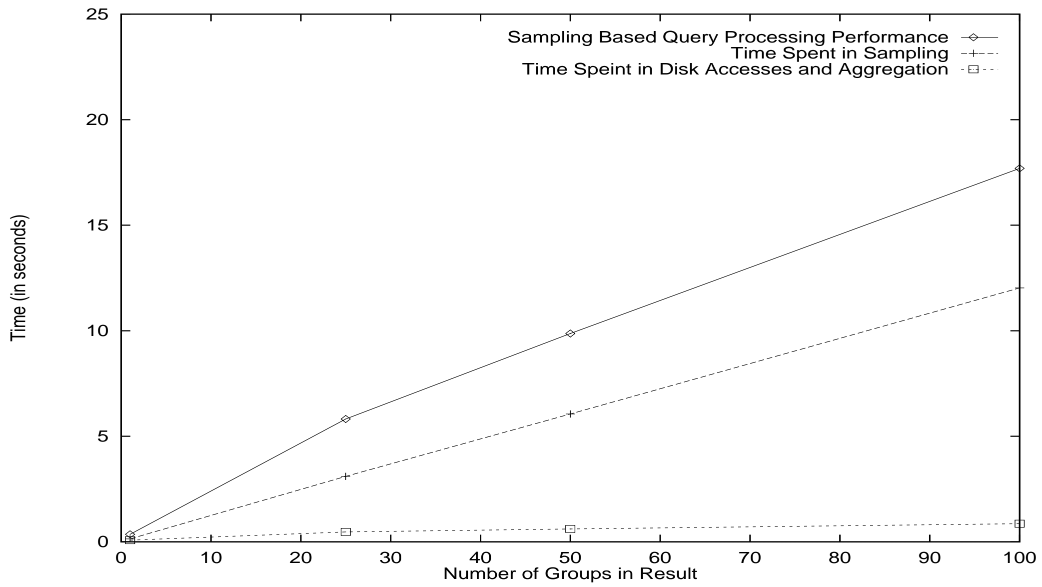

Figure 5: Effect of Number of Groups on Sampling Based Query Processing

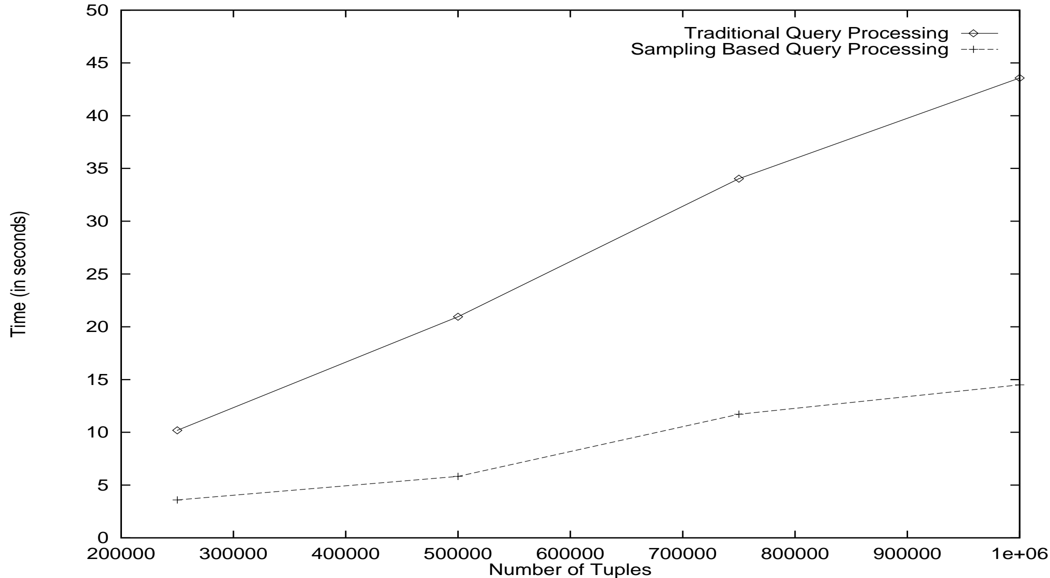

Figure 6: Effect of Number of Tuples on Overall Performance

main cost of query processing is in sampling. The cost of disk accesses is very low because sample sizes are typically small. Thus CPU overheads, instead of I/O overheads, dominate query processing in this case. The cost of sampling increases linearly with the number of groups because each group has to be sampled individually (see Section 2.2). This causes the performance of queries to deteriorate linearly with the number of groups. One effective way to speed up sampling based query processing would be to concentrate on efficient sampling techniques for group by clauses. We plan to explore this as part of future work (see Section 5).

4.3 Number of Tuples

Figure 6 compares the performance of traditional query processing with sampling based query processing when the number of tuples in the input relation is increased. The sampling based approach scales much better with increasing number of tuples. The reason for this can be seen from Figures 7 and 8.

In traditional query processing, the entire relation has to be scanned to answer a query. So, the number of disk accesses increases linearly with the size of the relation. In the sampling based approach, however, the number of disk accesses is not significantly affected by an increase in the number of tuples (though disk accesses over larger relations may be more expensive). This is because the sample size remains the same irrespective of the number of tuples. The cost of sampling also increases because the size of each bit map increases with an increase in the number of tuples. This increase, however, is not as dramatic as the increase due to increased disk accesses in traditional query processing.

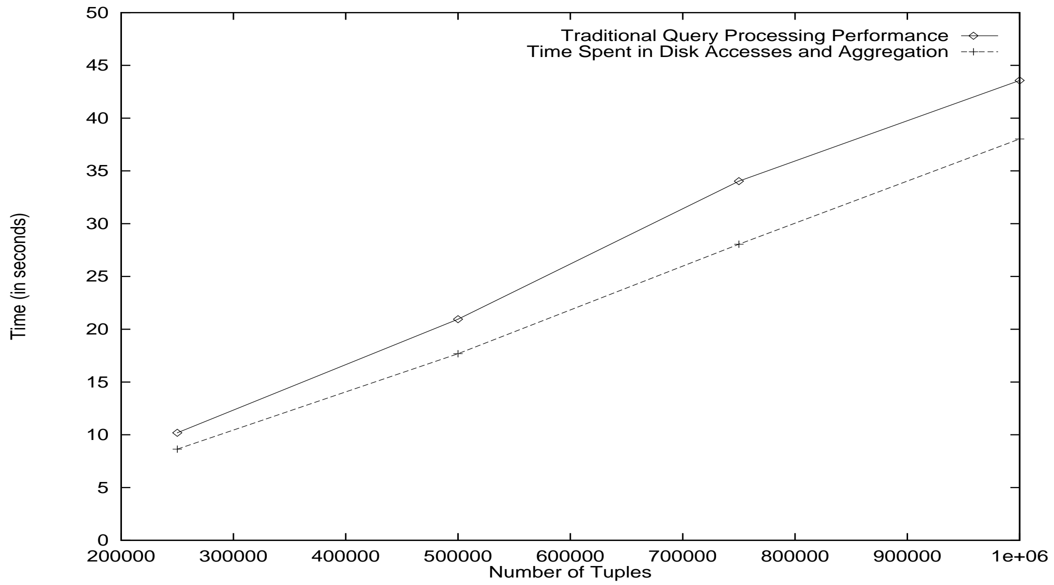

Figure 7: Effect of Number of Tuples on Traditional Query Processing

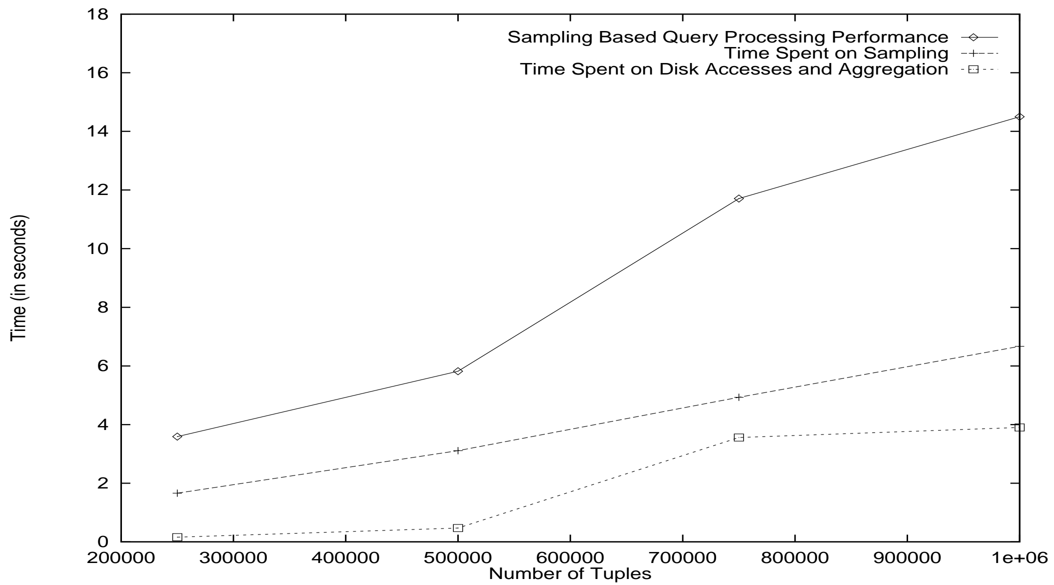

Figure 8: Effect of Number of Tuples on Sampling Based Query Processing

4.4 Bit Versus Byte Sampling Technique

Using a byte oriented approach to sampling as opposed to a bit oriented approach gives a factor of 2 improvement in sampling performance. Given that the costs of sampling are significant, especially when there are a large number of groups in the result, this could translate to significant savings. As a concrete example, for a query that produces 50 groups on a relation with 500000 tuples, this savings could amount to about 6 seconds, which would be a 40% savings in query execution time.

5 Conclusions and Future Work

In this report, we proposed an approximation based approach to OLAP query processing based on sampling bit map indices. We studied various sampling strategies and implemented two of them in a real database system. The performance results indicate that significant gains can be achieved using the new approach, especially when the number of groups in the results are not many. The new approach also scales much better than other approaches for large database sizes. Further, this techniques is easily integrated into existing database systems and can thus be of immense practical value.

There are plenty of open issues that remain to be considered. One of the potential concerns of the proposed approach is that the cost of sampling when there are a large number of groups may become prohibitively high that sampling may not be worth the effort. We envision two complementary approaches to this problem. One approach is to study techniques to speed up sampling, such as using alternative techniques, compressed bit map representations and sampling many groups at the same time. The other approach is to determine the conditions under which the proposed approach performs better than the existing approaches and based on this information, decide which approach is appropriate for (different parts of) a query. The sampling based approach also seems to be applicable in the context of tertiary storage systems, where the cost of I/O is far more than the cost of CPU computation.

Acknowledgment

We would like to thank Prasad M. Deshpande and Karthikeyan Ramasamy for their help during the during the course of this project.

References

[1] M. Carey, D. DeWitt, J. Naughton, M. Solomon, et. al. “Shoring Up Persistent Applications,” Proceedings of the SIGMOD Conference, Minneapolis, MN, 1994.

[2] W.G. Cochran, “Sampling Techniques,” John Wiley and Sons, 1960.

[3] D. DeWitt, N. Kabra, J. Luo, J.M. Patel, J. Yu, "Client-Server Paradise," Proceedings of the VLDB Conference, Santiago, Chile, 1994.

[4] V. Harinarayan, A. Rajaraman, J.D. Ullman, "Implementing Data Cubes Efficiently," Proceedings of the SIGMOD Conference, Montreal, Canada, 1996, pp. 205-227.

[5] P.E. O'Neil, D. Quass, "Improved Query Performance with Variant Indexes," Proceedings of the SIGMOD Conference, Tucson, Arizona, 1997, pp. 38-49.

[6] A. Shukla, P.M. Deshpande, J.F. Naughton, "Materialized View Selection for Multidimensional Datasets," (to appear) Proceedings of the VLDB Conference, Seattle, Washington, 1998.

[7] J.S. Vitter, "Faster Methods for Random Sampling," Communications of the ACM, vol. 27, no. 7, 1984, pp. 703-718.