Summary of "Cardinality Estimation Done Right: Index-Based Join Sampling"

By S. Xiao Fernández Marín

1 Summary

Cardinality Estimation is a hard thing to achieve, so in this paper, the authors try to solve this using sampling, executing a query with a limited sample of data set to measure the cardinality and using it as estimates for the entire query.

Sampling has not been used much in real systems as it is expensive when we can not find the data in the main memory and it has to run the optimizer twice. It also needs to use a large sample in order to get intermediate results.

A standard technique for estimating joins using sampling, where we find the **naive** and **index sampling**. If we have 3 samples (A, B and C), the former uses a join in A and B and then, that result is joined with C, which could end in an empty set. The latter uses an index-join between A and B and the same with that result and C. The result would be not as shrink as the former.

As seen previously, the index-based join sampling is better, as it is cheaper so it can compute more intermediate results. The number of these intermediate results grows exponentially so we need sampling, and it does that in a bottom-up enumeration. It improves the basis of the cost computations as it uses bread first (first 2-way joins, then 3-way,...). It also runs before optimization, it does not re-run. It samples 3-way joins as much, and the rest is estimated by samples, these estimates feed the optimizer and it limits the time spent in sampling.

For integrating the index-based join sampling into a traditional query optimizer, except for the addition of the index-based sample operator, there are no more changes. The plan produced by the optimizer is executed without additional execution time.

The algorithm that you can see in Figure 1, having a sample of a table A (with four rows) and joining it with an index B, produce some results (15) from the ones that it has counted. That number is reduced to 4 again in order to keep the sample size. We pick 4 random tuples and is joined with an index C and the same happens.

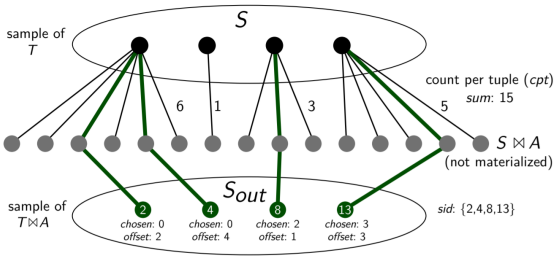

Figure 1: Algorithm

The algorithm counts the number of matches and does not compute an actual join result. It maintains the sample size, so if you need to generate a larger sample, just re-run it and reduce time. Also, it has cheap operations on indexes.

In the paper, they test it with the JOB data set and queries and they have a similar DB to MonetDB (in-memory, single-threaded and uses hash tables as indexes). It has a very simple cost model that can fit in the cardinality. They also inject cost estimates from PostgreSQL for comparison and they do it against sample-and-reoptimize strategy.

We can also find a plot of the quality of cardinality estimates for multi-join queries in comparison with the true cardinalities. In where there are 3 types, 10K (10K different samples), 100K and

with no budget (sampling as much as they want). It improves the longer they let it sample, but never perfect.

The problem of sampling comes when there is more than 8 join. If not, the budget limits the sampling. We can find the plot (Figure 2) of the overhead of sampling, we look at three lines where the cheapest one is the 10K (where when it reaches 4 it goes have the same cost and when it reach the 10K, it reaches budget and stops), the middle one the 100K and the expensive one, the no-budget one. We also have some circles that represent the sample-based re-optimization.

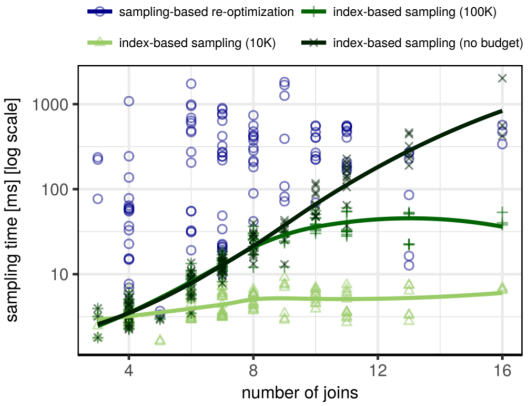

Figure 2: Overhead of sampling

Finally, we are shown different plots. The first one is the cost of the queries, where PostgreSQL is the worst and index-based sampling with no budget is the best. Then, actual runtime, where the results are the same as the previous one as they are very related. In the final plot, in the case we do not have indexes until 5 joins, we find that we do not find that much difference. Then we look at the plot that the PostgreSQL grows exponentially and the sample-based re-optimized scale up with the index-based sampling of 10K. The index-based sample of 100K and with no budget is doing better.

2 Questions not answered by this text

In the conclusion, they talk about integrating index-based join sampling into HyPer and investigating possible adaptations for DB with a larger RAM.

3 What has changed since this was written

The text is still valid in 2022, we are still using this even if there are some research about using machine learning.