Selectivity Computation for In-Memory Query Optimization

Jun Hyung Shin
University of California Merced
jshin33@ucmerced.edu

Florin Rusu
University of California Merced
frusu@ucmerced.edu

Alex Şuhan
MapD Technologies, Inc.
alex@mapd.com

Selectivity estimation for complex predicates. Consider the following SQL query:

```
SELECT R.A, S.B, R.C, R.D
FROM R, S, T
WHERE R.A = S.A AND S.B = T.B AND
   R.B = x AND R.C BETWEEN (y1, y2) AND
   (R.D = z1 OR R.D > z2) AND udf(R.B,R.C,R.D) > w
```

The tuples from table R that participate in the join are selected by a complex predicate $\sigma_{B,C,D}(R)$, over three attributes with exact, range, and OR conditions, and a user-defined function udf. When computing the optimal execution plan, i.e., join ordering, the query optimizer has to estimate the selectivity of $\sigma_{B,C,D}(R)$. When available, this is done with precomputed synopses, e.g., histograms, samples, sketches, stored in the metadata catalog. Otherwise, an arbitrary guess is used, e.g., for udf. Synopses are typically built for a single attribute and assume uniformity and/or independence when they are combined across multiple attributes. These are likely to miss correlations between attributes and result in inaccurate estimates which produce highly sub-optimal query execution plans.

In-memory databases. Database systems for modern computing architectures rely on extensive in-memory processing supported by massive multithread parallelism and vectorized instructions. GPUs represent the pinnacle of such architectures, harboring thousands of SMT threads which execute tens of vectorized SIMD instructions simultaneously. MapD (https://www.mapd.com/), Ocelot (https://bitbucket.org/msaecker/monetdb-opencl), and CoGaDB (http://cogadb.dfki.de/) are a few examples of modern in-memory databases with GPU support. They provide relational algebra operators and pipelines for GPU architectures that optimize memory access and bandwidth. However, they maintain the same synopses approach to query optimization as traditional disk-based databases.

Query optimization with exact selectivities. We introduce a novel query optimization paradigm for in-memory and GPU-accelerated databases—instead of estimating predicate selectivities, they are computed exactly through queries during the optimization. For the given query example, the optimizer instructs the execution engine to first perform the selectivity sub-query:

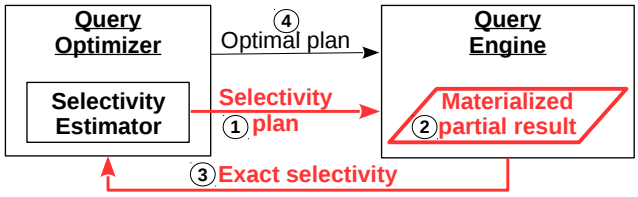

Figure 1: Exact selectivity computation.

```
SELECT R.A, R.C, R.D
FROM R
WHERE R.B = x AND R.C BETWEEN (y1, y2) AND
   (R.D = z1 OR R.D > z2) AND udf(R.B,R.C,R.D) > w
```

in order to compute the cardinality of $\sigma_{B,C,D}(R)$ exactly. This value is used together with the cardinalities of S and T to compute the best join order in the optimal query plan. Moreover, the result of the selectivity sub-query is temporarily materialized and reused instead of R in the optimal execution plan. The complete process is depicted in Figure 1.

While it is clear that the plan computed using exact selectivities is better – or at least as good – the impact on query execution time depends on the ratio between the execution time for the selectivity sub-query and the original plan. The assumption we make is that the sub-query execution is relatively negligible—valid for in-memory databases. We have to consider two cases. First, if the new query plan is improved by a larger margin than the sub-query time, the total execution time is reduced. We argue that exact selectivities are likely to achieve this for queries over many tables and with complex predicates. In the second case, the optimal query plan, i.e., join order, does not change even when selectivities are exact. Materialization minimizes the overhead incurred by the sub-query through subsequent reuse further up in the plan. In-memory databases prefer materialization over pipelining due to better cache access.

Our initial prototype implementation on top of the open-source MapD database proves the benefits of computing exact selectivities. We achieve up to 32X faster execution time for TPC-H scale 50 queries, while incurring at most 30 milliseconds overhead.

9th Biennial Conference on Innovative Data Systems Research (CIDR '19) January 13-16, 2019 , Asilomar, California, USA.