Bloom filter-aware Bottom-Up Cost Based Optimization

Tim Zeyl, Qi Cheng, Reza Pournaghi, Jason Lam, Ben Wang, Calvin Wong and Paul Larson

Cloud BU
Huawei Canada
timothyzeyl@huawei.com

1 Abstract

It is well known that Bloom filters can be applied in SQL query plans to perform early data reduction and that inclusion of Bloom filters can change the structure of the optimal query plan. To date, Bloom filter-aware query optimization has only been incorporated into top-down optimizers. We introduce a method to include Bloom filters in bottom-up cost-based optimization, applicable to all query types. We highlight the specific challenges that Bloom filters bring in limiting optimizer search space expansion, and we offer an efficient solution. We show that including Bloom filters in cost-based optimization can lead to better join orders with effective predicate transfer in a multithreaded database. On a 100G instance of the TPC-H dataset, we found a 32.8% reduction in latency for queries that involved Bloom filters. A version of this work was also presented at SIGMOD 2025[1].

2 BF-CBO

At a high level, our method (BF-CBO) adds additional sub-plans to scan nodes where Bloom filters can be used. The Bloom filters themselves will be created during Hash join building higher up in a join tree. These Bloom filter sub-plans at the scan nodes are costed and given reduced cardinality estimates. When evaluating joins between two relations, the additional Bloom filter sub-plans are combined with all compatible sub-plans from the other relation, and a unique cost and cardinality estimate is computed for the combination. This enables a different query plan to be found by the bottom-up optimizer, relative to a post-processing application of Bloom filters.

However, evaluating additional sub-plans necessarily increases the optimizer search space. Additionally, we show that Bloom filter sub-plans have a unique requirement that they cannot be pruned until the build-side of the Bloom filter appears in the partial join tree. That is because their row counts cannot be estimated until the complete set of tables on the build side of the join that provides the Bloom filter is known. If handled in a naïve way, the extra Bloom filter sub-plans must be maintained with *unknown* cost across several levels of planning. This naïve approach leads to an explosion of the search space and an increase in optimization time that is prohibitive.

Instead, our method is a two-phased bottom up approach to cost-based optimization, which allows us to delay planning until pruning is possible. In the first bottom up phase, we decide all valid join combinations, and identify all sets of relations that appear on the build-side of a join with Bloom filter candidate relations. This allows us to create costed Bloom filter sub-plans at the scan level with known cardinality estimates, these Bloom filter sub-plans carry with them an additional property of the set of relations required for resolution. We use heuristics to decide which scan-level Bloom filter sub-plans to try, and because we have full cost and cardinality information, we can also prune away bad sub-plans based on cost. In the second bottom up phase, we perform cost based optimization as usual, but with additional BF sub-plans adherent to certain combination restrictions.

3 Findings

Our method can find query plans with more Bloom filters applied and better predicate transfer. We tested on a 100G TPC-H database and found that end-to-end query latency reduced by 32.8% compared to the usual method of applying Bloom filters as a post-processing step - after cost-based optimization is finished. Planning time increased as expected, but increases were acceptable in our analytical query processing setting.

References

[1] Tim Zeyl, Qi Cheng, Reza Pournaghi, Jason Lam, Weicheng Wang, Calvin Wong, Chong Chen, and Per-Ake Larson. Including Bloom filters in bottom-up optimization. In *Companion of the 2025 International Conference on Management of Data*, SIGMOD/PODS '25, page 703–715, New York, NY, USA, 2025. Association for Computing Machinery. doi: 10.1145/3722212.3724440.