EFFICIENT COST BASED REWRITE IN A BOTTOM UP OPTIMIZER

Qi Cheng, Yang Sun, Weidong Yu, Yuanxi Chen, Weicheng Wang, Chong Chen, Paul Larson

Toronto Distributed Scheduling and Data Engine Lab(2012 Laboratories)
Huawei Technologies Canada Co., Ltd.

1 Introduction

A typical modern query optimizer relies on two core components: Query Rewrite (QRW) and Cost-Based Optimizer (CBO). The QRW applies logical transformation rules, such as predicate pushdown or subquery-to-join conversion, to restructure a query into a more efficient equivalent form. Subsequently, the CBO generates alternative physical execution plans, estimates their costs and selects the most efficient. Query rewrites are driven by predefined rules. A subset of these rules, known as heuristic rewrite rules, are always applied because they unconditionally improve performance. However, another class, known as cost-based rewrite rules, challenges this clean separation between the QRW and CBO phases. A cost-based rewrite is only beneficial if it leads to a lower-cost execution plan. Consequently, the optimizer cannot decide whether to apply such a rule during the QRW phase without consulting the CBO to evaluate the cost implications.

This architectural dilemma introduces significant computational overhead. For each candidate cost-based rewrite, the optimizer must pause the QRW process and invoke the CBO to compare the estimated costs of the original and rewritten queries. When multiple such rules are considered, these CBO invocations dramatically increases optimization time, in some cases rendering it impractical.

To address this well-known efficiency challenge, we propose a novel framework for a bottom-up optimizer. Our approach leverages an educated guess, derived from information available during the QRW phase, to establish a tight cost upper bound. This bound intelligently prunes the search space, guiding the optimizer toward better plans earlier in the process. Furthermore, the framework implements a multi-level caching mechanism for intermediate CBO plan results, at the base table, join, and subquery levels, which eliminates redundant computations across multiple CBO invocations. In this talk, we will present our solution and discuss our result.

2 Our Framework

Our approach is grounded in two key observations:

1) The benefit scenarios for a cost-based rewrite rule can often be identified using query characteristics and statistics which are available during the QRW phase without invoking the CBO.

2) Intermediate planning results, including base table accesses, joins, and untouched subqueries, are likely reusable across multiple CBO invocations for the same query.

During the QRW phase, when evaluating a cost-based rewrite rule, our method employs an "educated guess" based on available statistics, table physical properties, system resource, and query patterns to predict whether applying the rule will be beneficial. This guess is fast, as it avoids CBO invocation, and is likely accurate due to our deep understanding of the rule's benefit scenarios. As a safeguard, the CBO still evaluates both scenarios (applying the rule versus not applying it). However, the educated guess directs us to first compute the cost of the more promising alternative and establishes a tight upper bound that can allow early termination during the subsequent evaluation of the less promising option. This upper bound is continuously updated as lower-cost plans are found.

During the CBO phase, our framework caches plans for both query blocks and intermediate planning results. If a query block is unmodified by a rewrite rule, its cached plan is reused. For altered structures, the system maximizes reuse by leveraging cached intermediate planning results at multiple levels of granularity. The details of the framework, examples and various challenging scenarios will be elaborated in this presentation.

3 Result

The framework implemented in GaussDB was evaluated by measuring the compilation time of the 12 TPC-H queries that trigger cost-based rewrite rules. The results demonstrate an average improvement of 48% for cost-based rewrites compilation time. Since GaussDB currently supports only seven rules, expanding the rule set should yield even greater performance gains. In summary, this framework offers a more efficient methodology for applying cost-based rewrite rules, thereby significantly reducing the computational barrier to integrating a larger number of them into the query optimizer.