B. Mitschang et al. (Hrsg.): Datenbanksysteme für Business, Technologie und Web (BTW 2017),
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2017 25

Query Optimization – Are We There Yet?

Guy Lohman[1]

Abstract

After nearly 4 decades and hundreds of scientific papers, relational query optimization can hardly be characterized as anything but a huge scientific and commercial success. The market in 2016 for relational database products was estimated by IDC to be about $40B, out of a total database market of $45.1B. And SQL still dominates database application development and is widely recognized as the most successful declarative language. None of this would have been possible without the success of query optimization, which transforms declarative SQL statements of what data the user needs into an "optimal" ***execution plan***, i.e., a detailed, procedural specification for how that data will be accessed and processed.

So are we "there" yet? Are we done? Are all the big and interesting problems solved? Is query optimization as an area of scientific inquiry dead, relegated to incremental improvements and mere engineering? Why do we continue see so many papers on query optimization?

In this talk, I argue that current research appears to be incremental because we are largely attacking the wrong problems while ignoring much harder and more significant problems. We are solving the problems we know how to solve, not the problems that need solving.

Query optimizers are mathematical models of the performance of alternative plans. Any such model that is based upon invalid assumptions or that is not systematically validated throughout its parametric space is not worth the paper on which it is written, because it will inevitably yield wrong results at unknown points in that space. Current commercial optimizers are still largely dependent upon some simplifying assumptions made by the pioneers of query optimization, assumptions that too often are invalid. Yet these optimizers largely get decent plans most of the time because they luckily aren't near the break points between competing plans. We just don't know how bad it really is, because we debug optimizers by exception — that is, when we get an unexpected plan, or a customer complains — rather than by systematic and thorough validation.

Many of these remaining problems caused by invalid assumptions are contained in the Achilles Heel of query optimization: the underlying and ubiquitous ***cardinality model***, which estimates the number of rows resulting from each operation in the execution plan. Examples include the assumptions that constants in predicates are known at optimization time, that join domains enjoy typical key/foreign-key relationships of inclusion, and especially that predicates on columns are probabilistically independent. Additionally, traditional query

[1] BM Almaden Research Center (Retired), guy_lohman@alumni.pomona.edu

optimization cost models assumed that each query runs in isolation from other queries and focussed almost exclusively on the cost of magnetic disk I/Os, the "800-pound gorilla" of early optimizers. But recent advances in large main memories, flash storage, multi-core processors, and highly parallel in-memory database systems necessitate more accurate modeling of all these aspects (simultaneously!). Add to these the challenges of modeling non-relational operations (i.e., user- defined functions on steroids) and data types such as the arrays, repeating groups, and varying schemas of XML and JSON data types, common in Hadoop and now Spark applications, and you have an extensive research agenda.

Paradoxically, increasing the detail of optimizer models in response to these challenges may actually increase the brittleness of an optimizer! This happens because more detailed models inherently incorporate additional assumptions that may be invalid. Yikes! What is a conscientious query optimizer guru to do? I argue that robust and adaptable query plans are superior to optimal ones, that the goal of query optimization is more to avoid the occasionally really bad plan than to ensure the optimal plan, a process that Bruce Lindsay dubbed "goodizing". Accordingly, optimizers should substitute known facts for models whenever possible, an insight that spawned our idea of a "LEarning Optimizer" (LEO).

I will illustrate these problems, and a few possible solutions, with examples and "war stories".