Making Learned Query Optimization Practical: A Technical Perspective

Volker Markl
Technische Universität Berlin, Germany
volker.markl@tu-berlin.de

Query optimization has been a challenging problem ever since the relational data model had been proposed. The role of the query optimizer in a database system is to compute an execution plan for a (relational) query expression comprised of physical operators whose implementations correspond to the operations of the (relational) algebra. There are many degrees of freedom for selecting a physical plan, in particular due to the laws of associativity, commutativity, and distributivity among the operators in the (relational) algebra, which necessitates our taking the order of operations into consideration. In addition, there are many alternative access paths to a dataset and a multitude of physical implementations for operations, such as relational joins (e.g., merge-join, nested-loop join, hash-join). Thus, when seeking to determine the best (or even a sufficiently good) execution plan there is a huge search space.

Query optimizers use a cost model to guide this search, which is usually a linear combination of the cardinality (i.e., the expected size of an intermediate result that needs to be computed when processing a query) and physical parameters, such as the I/O transfer cost and CPU processing cost. While the I/O transfer cost and CPU processing cost can easily be measured, it is very hard to accurately model the size of an intermediate result that may arise during query processing, in particular after many joins, selections, or projections with duplicate elimination have been applied to a data set.

Traditionally, query optimizers rely on statistics about a table and the database in conjunction with assumptions, such as the independence of predicates occurring in selections, the inclusion of one data set in another data set for key-foreign key joins, or uniformity when more informative statistics are unavailable. When these assumptions are invalid or the statistical information available is outdated or missing altogether, the cardinality estimates produced by the cost model of the query optimizer may be off, often by orders of magnitude. This in turn is the major cause for poor execution plans and suboptimal performance during query processing.

This problem has been well recognized by the research community, and in the early mid-90s researchers started to look into learning methods for specific statistical artefacts. In particular, to monitor cardinalities during query execution and exploit the learned information in future query compilations. Chen and Roussopulos [2] used query results to learn the statistical distributions of simple predicates after the execution of a query and accordingly adapt the coefficients via a curve-fitting function. Aboulnaga and Chaudhuri [1] introduced a query feedback loop, where the actual cardinalities monitored during execution are used to correct histograms.

LEO [4] pioneered the concept of a learning query optimizer, where a feedback loop is used to determine and adjust for discrepancies among cardinalities and other parameters of the cost model during all steps of the query execution plan. Recently, due to advances in machine learning and the emergence of Software 2.0, these ideas have received renewed interest. In particular, several works have proposed to use deep learning or reinforcement learning to compute the cardinalities or the entire cost model.

The Bao paper [3] is a consequent continuation of this line of research, which can take advantage of the prior knowledge hard-coded in the cost model of existing query optimizers, while leveraging the core ideas of learned optimizers (e.g., incorporating query feedback, learning from mistakes). BaoŠs key innovations include leveraging the existing query optimizer, using a deep convolutional neural network to predict the performance of queries and suggest hints, and leveraging Thompson sampling for plan selection. This approach allows Bao to be easily integrated and combined with existing systems, while being able to adjust to changes in the workload, data, and schema.

1. REFERENCES

[1] A. Aboulnaga and S. Chaudhuri. Self-tuning histograms: Building histograms without looking at data. In *SIGMOD*, 1999.

[2] C. M. Chen and N. Roussopoulos. Adaptive selectivity estimation using query feedback. In *SIGMOD*, 1994.

[3] R. Marcus, P. Negi, H. Mao, N. Tatbul, M. Alizadeh, and T. Kraska. Bao: Making learned query optimization practical. In *SIGMOD*, 2021.

[4] M. Stillger, G. M. Lohman, V. Markl, and M. Kandil. Leo - db2's learning optimizer. In *VLDB*, 2001.