APPLICATIONS OF MACHINE LEARNING IN QUERY EXECUTION OF DATABASE SYSTEMS

Zoran Babović[1,2], Filip Hadžić[3]

[1] Faculty of Engineering, University of Kragujevac, Sestre Janjić 6, 34000 Kragujevac, Serbia
e-mail: zbabovic@uni.kg.ac.rs
[2] Innovation Center of the School of Electrical Engineering, Bulevar Kralja Aleksandra 73, 11000 Belgrade, Serbia
[3] School of Electrical Engineering, University of Belgrade, Bulevar Kralja Aleksandra 73, 11000 Belgrade, Serbia
e-mail: hadzic.filip@etf.rs

Abstract:

In recent years there has been a huge increase in data volume which database management systems (DBMS) have to efficiently manage by providing analytical functions as well as providing online transactional processing (OLTP) for various applications. The most challenging task in the DBMS is the efficient query execution which is the responsibility of a component called the query optimizer. Several benchmarks have shown that there is a lot of room for the improvement of existing query optimizers. The most recent research efforts in this field are substantially impacted by the advances in the field of machine learning (ML), which resulted in the application of various ML algorithms for improving the performance of query optimizers. This paper gives a brief overview of the approaches that apply ML algorithms at different levels within the query execution engine, including index structures, cardinality estimation, query enumeration plan, controlling existing query optimizer, and database parameters tuning. In addition, the main challenges for applying ML algorithms in DBMS are also discussed.

Keywords: query optimizer, cardinality estimation, deep learning, learned index structures

1. Introduction

Researchers have been working on the problem of query optimization for several decades [1]. In order to test the efficiency of query optimization, benchmarks such as the Join Ordering Benchmark (JOB) [2] have been created which show that sometimes DBMS query optimizers make dramatic mistakes when creating a query execution plan, thus unnecessarily increasing query execution time by several orders of magnitude.

The first step during query execution is cardinality estimation which is to determine the size of the intermediate results of a query. Based on the cardinality estimation, a cost model is generated for various types of operators that can be applied during the execution of subqueries, i.e. table join operations, such as nested loop join, hash-join, and sort-merge join. The efficiency of these operators depends significantly on the characteristics of the selected data over which they are executed. In the end, based on the cost model the optimizer selects, the most efficient query enumeration plan which is the order in which the operators are executed. The most enormous error in the total query execution time could cause an incorrect cardinality estimation. So, it is clear how important it is that cardinality estimation should be made as accurately as possible.

In addition, the query execution time could be significantly improved by reducing the data access time of auxiliary index structures, which are used by given join operators, e.g. hash-

join. Standard index structures used in DBMS are based on B-Trees, and there may be special structures for spatial data or multi-dimensional data based on variation of R-Tree or QuadTree.

Research efforts that investigate the application of ML algorithms to improve query execution are turned toward improving some of the aforementioned components of DBMS. In the next section, we will give a short overview of such approaches.

2. Classification of ML Algorithms Applied on Query Execution

Existing research efforts with applications of ML algorithms target the query execution at the different levels ranging from low-level index structures to high-level DBMS tuning through adjusting selected working parameters. We identified approaches that could be classified into several categories: approaches targeting low-level index structures, approaches that focus on the cardinality estimation task, approaches dealing with query enumeration plan, approaches that control the existing DBMS's query optimizer by sitting on top of it, and approaches that perform automatic database parameters tuning.

2.1 Index structures

Substantial advancements are achieved by the implementation of deep learning-based indexes to enable efficient low-level data access and searches in read-only databases [3][4] in order to replace B+ Trees indexes. This approach uses a hierarchy of models in which each model represents the node within a B+ Tree. The model is created by learning the cumulative distribution function (CDF) from the indexed data. The leaf node provides the position of the key within the sorted array. The solution outperforms the B+ Tree by a factor of 3 in respect of search time, and also has an order of magnitude lower memory consumption. A further application of the basic idea is used for multi-dimensional indexes which are exploited in analytical databases [5]. Both approaches could only be used in read-only databases, i.e. for OLAP types of workloads, whereas the further improvement of updatable learned indexes with the support of insert, update, and delete operations on data is given in [6] and it also is suitable for use with OLTP workloads.

2.2 Cardinality estimation

The most recognizable approach to the problem of cardinality estimation was done by Kipf et al. [7] and it is based on supervised deep-neural networksby utilizing multi-set convolutional neural networks. In this approach, a query is represented as a feature vector consisting of a table, join, and predicate data sets which are subsets of all available tables, joins, and predicates respectively. For example, table set data is a bit-vector where non-zero entry on a certain position within the vector denotes that a specific table is referred in the query. In the predicate features, data values are normalized to the range [0, 1] from minimum and maximum values. Evaluation is performed with the IMDB dataset and it shows that this approach makes a significantly lower estimation error, in some cases up to two orders of magnitude, compared to the PostgreSQL estimator and two other popular traditional approaches.

Other approaches use several deep autoregressive models [8]. On the other hand, Wang et al. [9] conducted a study in which seven learning methods for cardinality estimation were evaluated. The authors conclude that although these learning methods have better accuracy than the traditional methods, they are not ready for real production systems because of the training time and inference delay, as well as poor behavior with the frequent data updates.

2.3 Query enumeration plan

Krishnan et al. [10] addressed the query enumeration plan by exploiting deep reinforcement learning, whereas some authors [11] put efforts to predict query performance. Unlike applications of deep learning models, some approaches employ traditional machine

learning algorithms [12] such as linear regression, decision trees etc. for cardinality learning and query optimizer.

2.4 Controlling existing query optimizers

The approach implemented in Bao [13] utilizes the existing query optimizer in DBMS and it only activates a particular feature or operator (e.g. join algorithm) for a certain class of queries. Bao is based on the reinforcement learning algorithm with a tree convolution neural network together with Thomson sampling. This allows Bao to learn from the incorrect query plans and to adjust to changes in the workload and data. A typical situation is when the learned optimizer provides hints for applying a different join algorithm, i.e. nested loop join, hash join, or sort-merge join, from the algorithm the original query optimizer has selected. The variation of this approach is to activate the learned optimizer only for those types of queries for which the original optimizer does not give good performance. However, this approach depends on the ability to provide hints to the existing query optimizer of a particular DBMS which varies significantly among available DBMS.

2.5 Database parameters tuning

A pioneering approach with ML based DBMS tuning is the project OtterTune [14], which consists of components that execute the workload on the DBMS, collects performance metrics and configuration knobs from the DBMS, and stores them in the repository. In addition, performance metrics from previous sessions can also be utilized. Several ML models are then used for creating configuration recommendations by tuning particular knobs. In the experiment with issue-tracking production-level application running on Oracle DBMS, three ML algorithms are compared: Gaussian Process Regression (GPR), Deep Neural Network (DNN), and Deep Deterministic Policy Gradient (DDPG) where the best performance improvement was 45%, achieved by DNN. However, this result depends on the number of tuning knobs and the DBA assistance in selecting knobs. Other examples are CDBTune [15] and QTune [16] which are based on the deep reinforcement learning algorithm and utilize reward function for providing performance data feedback and also use DDPG method for setting knobs.

3. Challenges for Applying ML algorithms for Query Execution

Here we will list some of potential issues that could limit the applicability of machine learning algorithms for query execution in DBMS:

Initial training samples – Supervised learning techniques require initial training data which are sometimes difficult to acquire as in the case of collecting true cardinality estimation. Also, reinforcement learning requires a lot of query examples in order to improve performance of the query optimizer. The approach in [7] improves training quality by generating additional training examples which cover some workloads with higher error distributions.

Model Generality - Some models are applicable only when certain conditions are fulfilled, for instance, the constraints which were present during the training of the model. Some cardinality estimation models are applicable to general queries as well as to queries with complex predicates if they are trained with a limited set of queries.

Model adaptability – Training the model always requires some time and computational resources. If it is necessary to re-train the model, will that hurt the performance and induce some additional delay in query execution? As a result, training the model might be allowed only overnight.

Balanced performance – In most cases, i.e. in average, approaches with ML algorithms achieve better results than the traditional algorithms However, there are outlier cases when learned models have huge response times and delays unacceptable for real systems. This can prevent the use of such approaches in production systems.

4. Conclusions

This paper provides a brief overview of ML used for improving query execution in DBMS. Although the existing approaches are still not ready for production level use, most of the described approaches are very promising.

References

[1] G. Lohman, „Is Query Optimization a ‘"Solved" Problem?“, In ACM SIGMOD Blog, ACM Blog ’14, 2014.

[2] V. Leis, A. Gubichev, A. Mirchev, P. Boncz, A. Kemper, and T. Neumann, „How Good Are Query Optimizers, Really?,“ PVLDB, 9(3):204–215, 2015.

[3] T. Kraska, A. Beutel, E. H. Chi, J. Dean, N. Polyzotis, "The Case for Learned Index Structures," In Proceedings of the 2018 International Conference on Management of Data, ACM SIGMOD '18, pp. 489-504, 2018.

[4] T. Kraska, et al,"SageDB: A Learned Database System," Conference on Innovative Data Systems Research (CIDR), 2019.

[5] V. Nathan, J. Ding, M. Alizadeh, and T. Kraska, "Learning Multi-dimensional Indexes," In Proceedings of the 2020 International Conference on Management of Data, ACM SIGMOD '20, 2020.

[6] J. Ding et al., "Alex: An updatable adaptive learned index," In Proceedings of the 2020 International Conference on Management of Data, ACM SIGMOD ’20, pp. 969–984, 2020.

[7] A. Kipf, T. Kipf, B. Radke, V. Leis, P. A. Boncz, A. Kemper, "Learned Cardinalities: Estimating Correlated Joins with Deep Learning," In CIDR 2019.

[8] Z. Yang et al.,”Deep unsupervised cardinality estimation,” In PVLDB, 13(3), 2019.

[9] X. Wang et al., "Are We Ready For Learned Cardinality Estimation?," PVLDB, 14(9):1640-1654, 2021.

[10] S. Krishnan, Z. Yang, K. Goldberg, J. M. Hellerstein, and I. Stoica, "Learning to optimize join queries with deep reinforcement learning," CoRR, abs/1808.03196, 2018.

[11] R. Marcus, P. Negi, H. Mao, C. Zhang, M. Alizadeh, T. Kraska, O. Papaemmanouil, N. Tatbul (2019). Neo: a learned query optimizer," In Proceedings of the VLDB Endowment, 12(11), July 2019.

[12] C. Wu, A. Jindal, S. Amizadeh, H. Patel, W. Le, S. Qiao, S. Rao. Towards a Learning Optimizer for Shared Clouds. PVLDB, 12(3): 210-222, 2018.

[13] R. Marcus et al., "Bao: Making Learned Query Optimization Practical," In Proceedings of the 2021 International Conference on Management of Data (SIGMOD ’21), 2021.

[14] D. V. Aken, D. Yang, S. Brillard, A. Fiorino, B. Zhang, C. Billian, and A. Pavlo, "An Inquiry into Machine Learning-based Automatic Configuration Tuning Services on Real-World Database Management Systems," Proc. VLDB Endow. 14(7):1241-1253, 2021.

[15] G. Li, X. Zhou, S. Li, and B. Gao, „QTune: A Query-Aware Database Tuning System with Deep Reinforcement Learning, Proc. VLDB Endow. 12(12):2118–2130, 2019.

[16] J. Zhang et al. “An end-to-end automatic cloud database tuning system using deep reinforcement learning,” In SIGMOD, 415–432, 2019.