External vs. Internal: An Essay on Machine Learning Agents for Autonomous Database Management Systems

Fatima Khalil Aljwari[1†]
fatima6794o5@gmail.com
Umm Al Qura University
Mecca, Saudi Arabia

Summary
There are many possible ways to configure database management systems (DBMSs) have challenging to manage and set.The problem increased in large-scale deployments with thousands or millions of individual DBMS that each have their setting requirements. Recent research has explored using machine learning-based (ML) agents to overcome this problem's automated tuning of DBMSs. These agents extract performance metrics and behavioral information from the DBMS and then train models with this data to select tuning actions that they predict will have the most benefit. This paper discusses two engineering approaches for integrating ML agents in a DBMS. The first is to build an external tuning controller that treats the DBMS as a black box. The second is to incorporate the ML agents natively in the DBMS's architecture.

Keywords:
Essay, Machine Learning, Database Management Systems, External, Internal.

1. Introduction

A-Tuning a DBMS is an essential part of any database application installation. These tuning aims to improve a DBMS's operations based on some function (e.g., faster execution, lower costs, better availability). Modern DBMSs provide Application Programming Interfaces (APIs) that allow database administrators (DBAs) to control their runtime execution and storage operations:(1) physical design, (2) knob configuration, (1) hardware resource allocation, and (4) query plan hints. The Physical design changes to the database's physical representation and data structures. The knob configuration is optimizations that affect the DBMS's behavior. Resource allocations determine how the DBMS uses its available hardware to store data and execute queries; the DBA can use either provision new resources or redistribute existing resources. Lastly, query plan tuning hints are directives that force the DBMS's optimizer to make decisions for individual queries. In the 1970s, the efforts were building self-adaptive systems [5]. These were primarily tools based on physical database design. In the 1990s, the database community transformed into self-tuning systems [12]. Most self-tuning systems targeted automated physical design [2].

The current research trend is using machine learning (ML) to devise learned methods for automated DBMS tuning. The newer approaches train models using data collected about the DBMS's runtime behavior under various execution scenarios and configurations. The agent observes the effects of the deployed action and integrates the new data into the models to improve their efficacy for future decision-making.

There are two techniques developers can integrate ML-based tuning methods for DBMSs. The first technique is to use external agents that observe and manipulate a DBMS through standard APIs, While the second technique is to integrate internal components inside the DBMS.

This research discusses the trade-offs between implementing ML-based tuning agents outside of the DBMS versus designing a new DBMS around automation. We start in Section background with an overview of how DBMS tuning algorithms work. Next, we describe how to use ML-based agents to tune systems automatically. Then we compare the benefits of the approaches and discuss their challenges and problems. Finally, we conclude with an overview of two DBMS projects [1] using ML for automated tuning. The first project is OtterTune [5], an external tuning service for existing DBMSs, While The second project is a new self-driving DBMS called NoisePage [1] designed to be completely autonomous.

2. Background

Previous researchers have studied and devised methods for automatically optimizing DBMSs. A large group of previous work includes theoretical and applied research [14].Before the 2010s, the methodologies for automated DBMS tuning were either heuristic- or cost-based optimization algorithms.ML is a large field of study that includes many disciplines and applies to many facets of DBMSs. We discussion to the Effects of integrating ML in DBMSs for tuning in either existing or new systems.

The most used approach for automated DBMS tuning is heuristic algorithms. While the other approach for DBMS tuning is to use cost-based algorithms that programmatically search for improvements to the DBMSs. The Previous work in the cost-based algorithm has evaluated several search techniques, including greedy search [2], branch-and-bound search [14, 8], local search

Manuscript received October 5, 2023
Manuscript revised October 20, 2023
https://doi.org/10.22937/IJCSNS.2023.23.10.20

[13], and genetic algorithms [8].The most notable application of this optimization was Microsoft's Auto Admin application use of SQL Server's built-in cost models.

3. Automated Tuning with Machine Learning

ML-based agents for automated DBMS tuning use the algorithms dependent on models to select actions that improve the system's target. The agent extracts patterns and inferences from the DBMS's past behavior to predict the expected behavior in the future and learn how to apply it to new actions. The agent selects an action that it believes will provide the most benefit to its target.

Agents build their models from training data from the DBMS and its environment. And the type of data that an agent collects from the DBMS depends on its action domain. Many agents objective a specific sub-system in the DBMS, and thus they need training data related to these parts of the system. How an agent acquires this data depends on whether it trains its models offline or online.

Note that offline versus online approaches are not mutually exclusive, and a DBMS can use them together.ML methods had divided into three broad categories: There are existing DBMS tuning agents that use either one class of algorithms or some combination. We now describe these approaches in the context of DBMS tuning: (1) supervised, (2) unsupervised, and (3) reinforcement learning.

a) Supervised Machine Learning:

The agent builds models from training data that contains both the input and expected output (i.e., labels). The goal is for the models to learn how to produce the correct answer for new inputs. This approach is helpful for problems where the outcome of an action is immediately observable. The training data input (e.g., type, predicates, input data sizes) and the output is the cardinality that the DBMS observed when executing the query. The objective of this agent is to minimize the difference between the predicted and actual cardinalities. Supervised learning has also been applied to tune other parts of a DBMS, including performance modeling [4].

b) Unsupervised Machine Learning:

With this approach, the agent's training data only contains input values and not the expected output. The agent doesn't need to know what "correct" values are to figure out what values do not look like the correct values .It is up to the agent to infer whether the output from the models is correct or not.

c) Reinforcement Machine Learning:

Lastly, reinforcement learning (RL) is like unsupervised ML in that there is no labeled training data. The agent trains a policy model that selects actions to improve the current environment's target objective function. RL approaches, in general, do not make assumptions about priors, and the models are derived only from the agent's observations. That is useful for problems where the benefit or effect of an action is not immediately known. For example, the agent may add an index to improve query performance, but the DBMS will take several minutes or even hours to build it. Given the general-purpose nature of RL, it is one of the most active areas of DBMS tuning research in the late 2010s. Researchers have applied RL for query optimization [7], index selection [10], partitioning [3, 6].

We next discuss how to integrate agents that use the above ML approaches into DBMSs to enable them to support autonomous tuning and optimization features. We begin with examining strategies for running agents outside of the DBMS in Section 3.1. Then in Section 3.2, we consider the implications of integrating the agents directly inside the DBMS. We first present the approach for each of these strategies and then list some of the challenges one must overcome.

3.1 External Agents:

An external agent tunes a DBMS without requiring specialized ML components running inside the system. The goal is to reuse the DBMS's existing APIs and environment data collection infrastructure without modifying the DBMS itself. Ideally, a developer can create the agent for a general-purpose such that one can reuse its backend ML component across multiple DBMSs. An agent receives objective function data directly from the DBMS or additional third-party monitoring tools.

Agents may also need an additional controller running on the same machine as the DBMS or within the same administrative domain [11]. This controller can install changes that are not accessible through the DBMS's APIs. The controller may also need to restart the DBMS because many systems cannot apply changes until restart.

Challenges:

The challenges did not design in tuning DBMS for autonomous operation. Foremost is that almost every major DBMS that we have examined does not support making changes to the system's configuration without periods of unavailability, either due to restarts or blocking execution [9].

Requiring the DBMS to halt execution to apply a change makes it difficult for agents to explore configurations and increases its time to collect training data.

The second issue is that an agent can only collect performance metrics that the system exposes.

Mean that if there is additional information that the agent needs, then it is not immediately available. The other issue is that data is often overabundance, making it challenging

to separate signals from the noise [11]. DBMSs also do not expose their underlying hardware to reuse training data across operating environments.

3.2 Internal Agents:

An alternative to treating the DBMS like a black box and tuning it with an external agent is to design its architecture to support autonomous operation natively. The DBMS supports one or more agents running inside the system with this approach. These agents collect their training data, apply changes to the DBMS, and then observe how they affect the objective. The system does not require guidance or approval from a human for any of these steps. The benefit of running agents inside of the DBMS is that it exposes more information about the system's runtime behavior and can potentially enable more low-level control of the DBMS than what is possible with an external agent. Most of the proposed ML tuning agents available today do design to extend or replace components in existing DBMSs.

Challenges:

The biggest problem with replacing a DBMS's existing components with new ML-based implementations is hard to capture their dependencies. Consider a tuning agent that controls the DBMS's memory allocations. Suppose the agent initially assigns a small amount of memory for query result caching and a large amount to the buffer pool. Another index tuning agent running inside the same DBMS then chooses to build an index because memory pressure in the buffer pool is low. But then the memory agent decides to increase the result cache size and decrease the buffer pool size. There are three ways to this problem, but each has its own set of issues. The first is to use a single centralized agent rather than separate agents. That is potentially the most practical but dramatically increases the complexity of the models, which requires significantly more training data.

The second is to have each agent provide a performance guarantee about its changes in the DBMS. The agents give this information to a central coordinator in charge of resource allocations. The last approach is a decentralized architecture where agents communicate and coordinate. One of the most expensive parts of these agents is when they build their models from the training data.

The controller is a conduit between the target DBMS and the tuning manager. It contains DBMS-specific code for collecting runtime information from the target DBMS (e.g., executing SQL commands) and installs configurations recommended by the tuning manager. The controller updates the DBMS's configuration file on disk and then restarts using the appropriate administrative tool.

The tuning manager updates its repository and internal ML models with the information provided by the controller and then recommends a new configuration for the user to try. The process continues even the user is satisfied with the improvements.

4.1 Machine Learning Pipeline

Otter Tune's ML pipeline uses a combination of supervised and unsupervised methods. It processes, analyzes, and builds models from the data in its repository. The Workload Characterization and Knob Identification modules execute a background task that periodically updates models as new data becomes available in the storage. The tuning manager uses these models to generate new knob configurations for the target DBMS. OtterTune's ML pipeline has three modules:

Workload Characterization:

This first component compresses all the past metric data in the repository into a smaller set of metrics that capture the distinguishing characteristics for different workloads.

Knob Identification:

The next component analyzes all past observations in the repository to determine which knobs impact the DBMS's performance for different workloads.

Automated Tuner:

In the last step, the tuner analyzes the results it has collected so far in the tuning session to decide which configuration to recommend next.

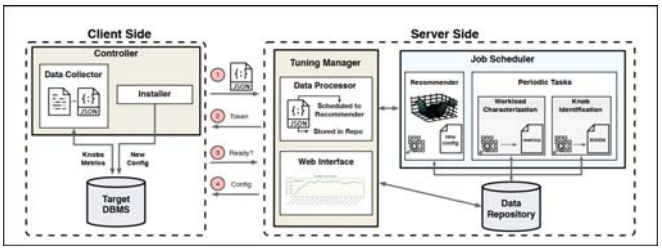

Fig 1: OtterTune Architecture

The controller connects to the target DBMS, then collects its data, transforms the collected information into a JSON document, and sends it to the server-side tuning manager; then it does the following:
Sends the information to the tuning manager.
Controller gets a token from the tuning manager.
Uses this token to check the status of the tuning job.
Gets the recommended configuration when the job finishes, and then the agent installs it in the DBMS.

5. NoisePage – (A Self-Driving) DBMS Architecture

NoisePage is a new DBMS that we are developing to be self-driving [1]. The DBMS's control agent runs in a continuous loop where it selects actions to deploy that it estimates will improve the target objective. That means that the system can tune and optimize itself automatically without any human intervention other than selecting the target function on start-up.
This (1) improves data collection efficiency and (2) reduces model over-fitting. The DBMS then combines this offline data with data collected online during query execution to improve accuracy.

5.1 Machine Learning Pipeline

We next provide an overview of Noise Page's self-driving pipeline. As shown in Figure2 illustrates the overall architecture of the DBMS with its modeling and planning modules.

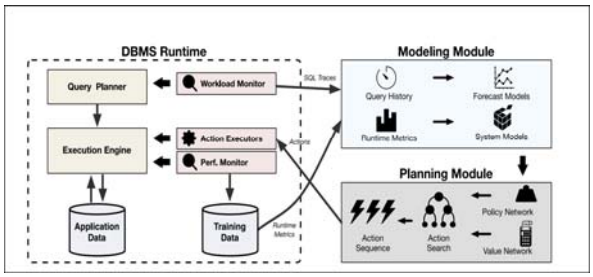

Fig 2: overall architecture of the DBMS with its modeling and planning modules.

Modeling: This first module is responsible for training prediction models using data that the monitors collect from observing its runtime process, such as forecast models that predict the application's future workload and database state. Forecasting is necessary so that DBMS can prepare itself accordingly, much like a self-driving car has to indicate the road condition up ahead.

Planning: In the second module, the DBMS uses the models generated in the previous step to select actions that provide the best reward given the system's current state.
Deployment: For a given action selected in the planning module, the next step is for the DBMS to deploy it.
The DBMS's planning modules use the data it collects during this phase to update their models and improve their decision-making.

6. Conclusion

This paper surveyed the approaches for adding automatic tuning agents based on ML to DBMSs. We discussed the high-level differences of external versus internal agents and the different challenges in these approaches. We then discussed two examples of architectures: OtterTune [5] and NoisePage [1]. However, more research in both systems and ML before achieving fully autonomous means that self-driving DBMSs. We want to make one final point: we believe autonomous DBMSs will not supplant DBAs. We instead envision these systems will emancipate them from low-level tuning and allow them to pursue higher- tasks, such as database design and development.

References

[1] NoisePage. https://noise.page.
[2] S. Chaudhuri and V. R. Narasayya. An efficient cost-driven index selection tool for Microsoft SQL Server. In VLDB, pages 146–155, 1997.
[3] G. C. Durand, M. Pinnecke, R. Piriyev, M. Mohsen, D. Broneske, G. Saake, M. S. Sekeran, F. Rodriguez,and L. Balami. Grid formation: Towards self-driven online data partitioning using reinforcement learning. aiDM'18, pages 1:1–1:7, 2018.
[4] A. Ganapathi, H. Kuno, U. Dayal, J. L. Wiener, A. Fox, M. Jordan, and D. Patterson. Predicting multiple metrics for queries: Better decisions enabled by machine learning. In International Conference on Data Engineering, pages 592–6017. IEEE, 2009.
[5] M. Hammer. Self-adaptive automatic database design. In National Computer Conference, AFIPS' 77,pages 123–129, 1977.
[6] B. Hilprecht, C. Binnig, and U. Rohm. Towards learning a partitioning advisor with deep reinforcement ¨learning. In aiDM@SIGMOD, pages 6:1–6:4, 2019.
[7] R. Marcus and O. Papaemmanouil. Towards a hands-free query optimizer through deep learning. In CIDR 2019, 9th Biennial Conference on Innovative Data Systems Research, 2019.
[8] R. Nehme and N. Bruno. Automated partitioning design in parallel database systems. In SIGMOD,SIGMOD, pages 1137–1148, 2011.
[9] A. Pavlo et al. Make Your Database Dream of Electric Sheep: Engineering for Self-Driving Operation.2019. Under Submission.

[10] [A. Sharma, F. M. Schuhknecht, and J. Dittrich. The case for automatic database administration using deep reinforcement learning. CoRR, abs/1801.05643, 2018.
[11] [D. Van Aken, A. Pavlo, G. J. Gordon, and B. Zhang. Automatic database management system tuning through large-scale machine learning. Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD '17, pages 1009–1024, 2017.
[12] G. Weikum, C. Hasse, A. Monkeberg, and P. Zabback. The COMFORT automatic tuning project. ¨ Information Systems, 19(5):381–432, July 1994.
[13] [B. Xi, Z. Liu, M. Raghavachari, C. H. Xia, and L. Zhang. A smart hill-climbing algorithm for application server configuration. In WWW, pages 287–296, 2004.
[14] [D. C. Zilio, J. Rao, S. Lightstone, G. Lohman, A. Storm, C. Garcia-Arellano, and S. Fadden. DB2 design advisor: integrated automatic physical database design. In VLDB, pages 1087–1097, 2004.

Fatimah K. Aljwari, Bachelor degree in computer science, working at Umm Al Qura University, living at Al-Qunfudhah, Mecca, Saudi Arabia.