Self-Driving Databases
Index Selection & Workload-driven Optimizataion

Group 40 - Xiaoyi Liu[1], Runqiu Fei[2],
Qingxuan Yang[3], Sunchuangyu Huang[4]

July 19, 2023

[1] Student ID: 1401212; Email: xiaoyi10@student.unimelb.edu.au
[2] Student ID: 1166093; Email: runqiuf@student.unimelb.edu.au
[3] Student ID: 1359534; Email: qingxuan1@student.unimelb.edu.au
[4] Student ID: 1118472; Email: sunchuangyuh@student.unimelb.edu.au

Abstract

Introduction

As the volume of data collected from diverse sources continues to surge, the traditional methods of data management have proven insufficient and burdensome for database administrators. To address this, the data industry has recognized the need for a advanced system design that operates autonomously, termed as 'self-driving' databases [17]. These sophisticated systems utilize advanced artificial intelligence and machine learning techniques to autonomously perform lower-level tasks [16], thereby reducing human interventions to a minimum. This not only lowers the Total Cost of Ownership (TCO) [9] but also alleviates responsibilities from Database Administrators (DBAs) [12].

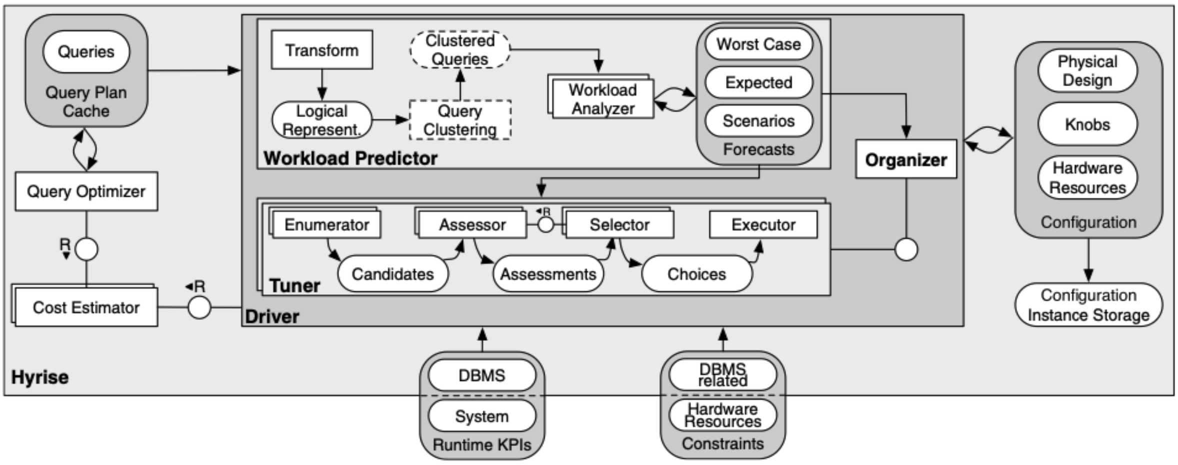


Figure 1: Conceptual self-driving database approach[9]

In 2020, Kossmann and Schlosser proposed a conceptual self-driving database structure, it comprised of three key components: the workload predictor, the tuner, and the organizer, as depicted in figure 1 [9]. Initially, the workload predictor estimates future workload, providing statistical data for the tuner. This tuner adjusts database configurations, also known as "knobs," to enhance system performance. The organizer, functioning as an orchestration tool, employs the predictions and tuning parameters to optimize database operations at the physical level. Consequently, a self-driving database aims to find the optimal workload management and effective knob configurations [9].

Level	Name	Description
0	Manual	The system has no autonomy.
1	Assistant	The system recommends promising actions to the user.
2	Mixed	The system performs actions and alerts the user for decision-making.
3	Local	The system has self-contained components that autonomously act.
4	Directed	The system is semi-autonomous.
5	Self-driving	The system is fully autonomous.

Table 1: Levels of Autonomy Capabilities [12]

Palvo et al., on the other hand, provide a granular view by defining levels of autonomy to evaluate whether a database management system is self-driving [16]. As present in table 1, self-driving database is defined with a level range from 0 (manual) to 5 (self-driving), depending on the degree of self-management and user/DBA involvement required [12].

In this report, we focus upon tuner and workload predictor components proposed by Kossmann and Schlosser [9] with a particular focus on *indexing selection* and *workload optimization*. Indexing is crucial in enhancing query performance and consequently the overall database efficiency [17]. Similarly, workload optimization help the system managing resources effectively, maximising the system performance with minimal latency [9].
—- need modification on the last paragraph, describe our approaches in two sections

Related Work - Workload-driven optimization

Overview

Previous studies have examined database workload modeling in different ways. Some focus on modeling resource demands within the system, while others analyze the DBMS performance by answering ”what-if” questions about changes in OLTP workloads. These components work together to enable proactive decision-making and resource allocation based on anticipated workload patterns. By integrating them, workload optimization makes database systems to operate autonomously and efficiently. It ensures that resources are allocated optimally, response times are minimized, and the system can adapt to changing workload demands.
Workload optimization plays a crucial role in self-driving database, which typically comprises three key components: workload forecasting, behavior modeling, and action planningenabling it to efficiently handle the diverse query patterns and evolving workloads encountered in real-world applications. By optimizing queries based on workload characteristics, a self-driving database can achieve improved performance and resource utilization.

1. Workload forecasting enables the system to predict future workload characteristics, such as query arrival rates and patterns. By analyzing historical data and employing machine learning techniques, it provides insights into future workload trends. This capability allows the system to proactively allocate resources, scale up or down, and efficiently manage the workload.

2. Behavior modeling focuses on capturing how the database system performs under different workload scenarios. By understanding the system's behavior, performance bottlenecks can be identified, system configurations can be evaluated, and the impact of workload changes can be assessed. This knowledge empowers system administrators to make informed decisions and optimize system performance.

3. Action planning involves selecting appropriate actions, such as adjusting resource allocation, optimizing query execution plans, or implementing caching mechanisms. By leveraging insights from workload forecasting and behavior modeling, it enables the system to dynamically adapt and optimize its operations based on anticipated workload conditions.

Following we will delve into each component, highlighting their importance and the benefits they bring to workload optimization in database systems.

Workload Forecasting

Workload forecasting involves predicting future workload characteristics. By accurately predicting future workload patterns, these systems can proactively allocate resources and optimize query execution strategies to meet service level agreements (SLAs) and enhance overall efficiency.

Resource Estimation: Some studies focus on automatically identifying workload trends and scaling resources for provisioning. These approaches use signals derived from DBMS internals to estimate resource demand and support short-term resource scaling. In contrast, QB5000 models query arrival rates for both short- and long-term horizons, enabling complex optimization planning decisions.

Performance Diagnosis: Prior research explores modelling and diagnosis of DBMS performance. For instance, DBSeer predicts the impact of workload changes on disk I/O by clustering transaction types and estimating resource utilization. However, these models primarily focus on fixed transaction types and do not consider future workload prediction. DBSherlock is a diagnostic tool that employs causal models to identify performance anomalies and assist DBAs in system analysis.

Shift Detection: Markov models have been used to predict the next SQL statement or transaction based on current DBMS activity. While some studies combine these models with workload classification techniques to capture periodic patterns, none of these methods effectively predict the volume, duration, and changes of future workloads.

The QueryBot 5000 (QB5000) framework, inspired by self-driving cars, presents an innovative approach to workload forecasting in DBMS deployments. Previous work in workload forecasting and modeling can be categorized into resource estimation, performance diagnosis, shift detection, and workload characterization as above.

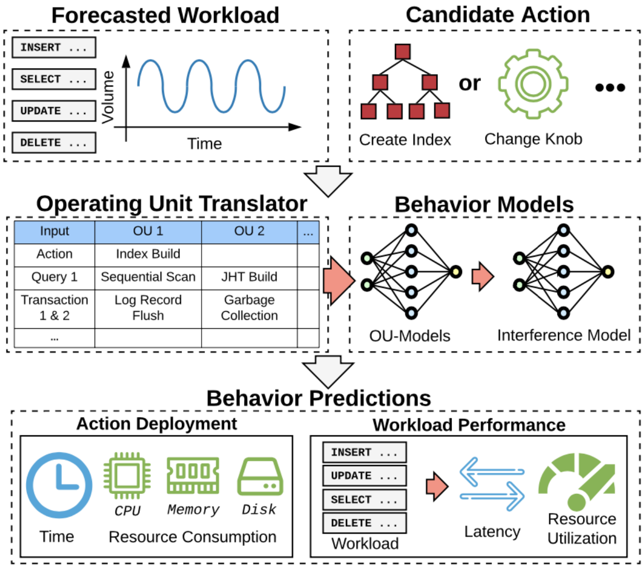
QB5000 addresses these limitations by providing accurate workload forecasts considering both short and long-term horizons. It leverages machine learning techniques, such as linear regression (LR), recurrent neural networks (RNN), and kernel regression (KR), to capture workload patterns and facilitate informed optimization decisions. It also offers a comprehensive approach to workload forecasting by addressing challenges related to arrival rate patterns, complexity and scalability, and workload evolution. By leveraging its architecture and machine learning techniques, workload predictions can be more accurate and dynamic, resulting in efficient resource allocation and improved performance of self-driving DBMSs.

Behavioural Modeling

Given the forecasted workload, the behaviour modeling process estimates and explains costs and benefits of all potential actions, laying foundations for self-driving decision-making.[13, 12] With every action, behaviour models predict various aspects including time duration, resource consumption, impact on system performance, etc.[13]

Previous research mostly fall into the following **two categories**[13, 12]:
Analytical models rely on human-devised, white-box formulas to describe the runtime behaviour of a DBMS, such as disk I/O, query time, buffer pool and lock contention[14, 20]. Various models developed targets different workloads and runtime components. Duggan et al.[8] and Ahmad et al.[4] focus on OLAP workloads while Resource Advisor[15], DBSeer[14], and DBSherlock[20] focus on concurrent OLTP workloads. Wu et al.[231, 232] estimate query execution time by leveraging optimiser's cost models. Despite their accuracy and efficiency, most models are customised for, and thus restricted to specific DBMS and workload(e.g. DBSeer with respect to MySQL[14]).

Machine learning models, in comparison, are far more adaptable. Of these, one subcategory specifically targets isolated query execution and uses query plan as input features[thesis]. Models such as hierarchical classification with PQR trees[90], subspace projection[83] and QPPNet with deep neural networks[141] all yield great accuracy. To improve generality, operational-level models are also incorporated. This includes runtime tuning of pre-built models[23] and implementation of additional scaling functions[133, thesis]. Another subcategory further models concurrent operations that predict performance for a set of queries. Representative models include Wu et al.[230] with queuing networks and Markov chains for dynamic query sets and GPredictor[249] with graph-based deep learning prediction network[thesis]. However, both subcategories could potentially incur expensive update and retraining processes upon new DBMS. With different synthetic benchmarks for different workloads, unstable prediction errors may be expected[thesis].



Input	OU 1	OU 2	...
Action	Index Build		
Query 1	Sequential Scan	JHT Build	
Transaction 1 & 2	Log Record Flush	Garbage Collection	
...			

Figure 4.3: MB2 Inference Procedure – Given the forecasted workload and a self-driving action, MB2 records the OUs for all input tasks, and uses the OU-models and interference models to predict the DBMS's behavior.

Figure 2: MB2[10]

With the above limitations and challenges in dealing with high input feature dimensionality, concurrent queries and internal DBMS operations, and model explainability, ModelBot2(MB2) is developed as a state-of-the-art holistic framework for in-memory DBMS[MB]. This framework decomposes DBMS into small, independent operating units(OU), whose corresponding OU-models are trained with data collected from relevant OU-runners. Overall inference models are built with end-to-end workloads data from concurrent runners. In this way, MB2 would be able to give a holistic behaviour prediction given forecasted workload and candidate action. Although MB2 achieves great accuracy, supports both OLAP and OLTP workloads, and could adapt to different DBMSs[MB2], a few limitations exist in predicting new queries, supporting disk-oriented DBMS with buffer pools, and using error-prone optimiser's cardinality estimations[130].

Action Planning

Action planning takes forecasted workload and behaviour model estimates as input and plans the best sequence of actions for the DMBS to apply at the correct timing[thesis]. This problem not only requires the model to solve a complex constrained optimisation in an exponential search space for the best sequence over a short or long horizon, but also needs to cater to system constraints and behaviour model's overhead.[thesis]

Previous research models could be divided into the following **two categories**[thesis]:
Static workloads models generates action for physical design or knob configurations given representative static workloads. Many models focus on optimised index configuration, such as Cophy[59], Kossman et al.[118], Das et al.[58], and DB2 Advisor[214]. Some target knob configurations, such as Aken et al.[215], Qtune[132], and Zhang et al.[244]. More recently, UDO uses reinforcement learning and Monte Carlo tree search(MCTS) to optimise transaction code, physical design, and system parameters[220]. Although they all give great optimisations given static workloads, a self-driving action planning framework needs to take into account temporal information in order to plan ahead for not only what to implement, but also how and when. GREEDY-SEQ[20] provides some partial solutions by using static workload tuning models to generate candidate actions, determine the best timing for each action by heuristics, and recursively merge them into a sequence. However, this model only tunes the physical design of a DBMS and may negatively impact optimisation process due to heuristics.

Online tuning methods target reactionary database tuning with shifting workload, solving real-time problems[thesis]. Bruno and Chaudhuri[37] continuously modifies physical design of the database in response to query workload, with candidates selected based on query execution statistics. DBA bandits[163] and COLT[185] focus on index creation and profiling resource allocation through analysis of workload and performance statistics. Although these methods all achieve great performance with real-time workload, they only solve problems after they occur and do not plan for future actions and workload patterns.

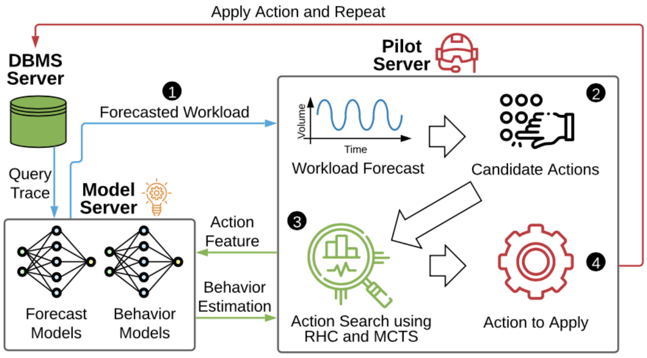


Figure 5.1: Pilot Architecture - PB0 runs the pilot server for planning on a separate machine from the DBMS server. It receives the forecasted workload from a model server to generate candidate self-driving actions. It then uses RHC and MCTS to search for the best action sequence based on the behavior estimations. It applies the first action and repeats the above process.

Figure 3: PB0[10]

Inspired by above models, Ma[thesis] developed a complete framework PilotBot0(PB0) for such self-driving action planning. As visualised in graph[], The framework is separate from DBMS and consists of a model server and a pilot server. Given an objective, the model server forecasts future workload patterns based on DBMS query trace. Then the pilot server generates a set of candidate actions using static workload methods, plans action sequences over a fixed horizon using receding horizon control(RHC) while repeatedly optimising the objective with Monte Carlo tree search(MCTS) and behavioural model. For every optimisation process, the first action of the sequence is applied to DBMS. Besides employing a two-level caching design for lower inference computational cost, it records workloads and explanations for debugging. As delicate as this framework could be in planning actions related to tuning indexes and knob values, it has a few limitations including the objective restricted to reducing average query latency, and the lack of models to tune complex features such as table partitioning and resource scaling[thesis].

Summary

Related Work - Index Selection

The Pivotal Role of Indexing in Self-driving Databases

Indexing stands as an integral element of database management systems, substantially augmenting the performance of queries. It establishes a superfluous structure that accelerates query execution, thereby emerging as an invaluable tool in optimizing system performance[3]. In the realm of self-driving databases, the autonomous system is tasked with the identification, selection, and implementation of fitting indexes that are grounded in a representative workload[21]. Without this workflow, producing indexes with inferior indexing quality, thereby hindering optimal system performance[18]. The selection of indexes necessitates a deep understanding of query plans and a continuous adaptation to workload changes, aligning seamlessly with the capabilities inherent in self-driving databases.

Indexing Selection Process

In general, the optimization process consists of three conceptual stages[3, 18, 21]:

1. Index Enumeration: Identify relevant indexes from workload data.
2. Candidate Selection: Remove unneeded indexes to enhance performance.
3. Exploration & Recommendation: Assess and suggest optimal index for configuration.

Consequently, the selection process involves the enumeration of potential indexes based on the query plan, followed by the removal of spurious or impractical indexing[3]. This is not a random process, but it necessitates the use of specific algorithms, such as the `Greedy` method, which suggests a configuration of indexes for a defined workload. The enumeration of potential indexes can result in an exponential outcome, and pinpointing the optimal physical design or ”knob configuration” from these possibilities is fundamental for performance enhancement[18]. The effectiveness of the chosen indexes is typically gauged using a ”what-if”[10, 9] approach, consolidating results sourced from the query optimizer[10].

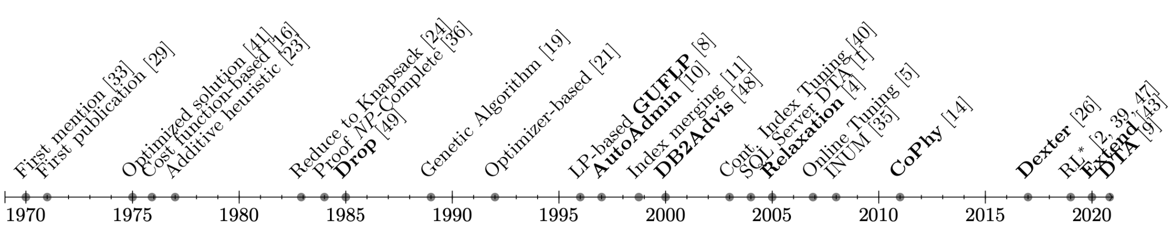


Figure 4: Timeline of milestones of index selection algorithms[10]

Over time, researchers have put forth several indexing selection and optimization algorithms, which can be grouped into three categories: heuristic (e.g. Drop[19]), linear constraint problem (e.g. AutoAdmin[6], CoPhy[7]), and machine learning approaches (e.g. DTA[5]). However, discovering an efficient index that minimizes workload cost presents a formidable challenge[10]. The evaluation metrics are complex, making a horizontal comparison of the algorithms difficult. Depending on the algorithm's characteristics and specializations, they operate in varying contexts. For example, *DTA*[5], *AutoAdmin*[3] trying to minimize the workload cost, meanwhile Dexter[10] aims to reduce the storage consumption. When confronted with a large-scale solution space, there is no 'state-of-the-art' algorithm for indexing selection underlining the challenges.

Indexing Selection Algorithms

The progression of indexing selection strategies began with heuristic techniques and gradually moved towards more intricate methodologies involving constraint problems and linear programming. As discussed by Ma et al.[12] and illustrated by Kossmann[10], researchers are transitioning from the traditional enumeration method towards contemporary techniques, such as leveraging linear constraint problems or utilizing reinforcement learning to discover the optimal index. In the following section, we will examine index selection algorithms from heuristic approaches to linear programming methods, and now, the application of machine learning techniques to automate index selection.

Inception: Heuristic Approaches

The heuristic approach was first introduced in 1985 with the DROP algorithm. This method aimed at efficiently eliminating indexes from index enumeration[19]. It focuses on the intrinsic data characteristics for evaluating cost performance rather than the optimizer. Consequently, the DROP algorithm could handle extensive selection on indexes, accommodating up to a maximum number of selected indexes denoted by `I-n`[19]. However, the method's heavy dependence on data attributes and lack of reliance on optimization techniques could lead to sub-optimal solutions.

In the year 2000, AutoAdmin with heuristic approaches, incorporating the application of constraints on feature spaces in indexing[3]. It is also a foundation of commercial applications like DAT[10, 5] developed by Microsoft. The method restricted the number of possible enumerations, extracting all feasible candidates within the given constraints. AutoAdmin employed a greedy approach to explore and evaluate index subsets efficiently, ensuring the process was performed within a reasonable time frame. Although this method reduced the runtime, it fell short in guaranteeing an optimal solution, especially when dealing with multi-column indexes[3].

Emergence of Constraint Problems and Linear Programming

Following the heuristic approach era, the focus shifted towards addressing constraint problems and leveraging linear programming for index selection. In 2000, Valentin et al. introduced DB2 Advisor, which marked a key milestone in this shift. This strategy involved the inclusion of hypothetical indexes (virtual indexes which do not exist physically) until a predefined quantity was met[18]. It found an optimal solution through the optimizer cost estimation, and then random variations of the index configuration were introduced to further refine the performance. However, its dependency on random alternations could sometimes yield the sub-optimal solution[18].

Building on these principles, the CoPhy algorithm was introduced in 2011 using an LP-based method to address the indexes optimization problem. In CoPhy, the goals (find and select optimal indexes) and constraints (budget) were expressed as linear equations, and the index selection problems were formulated as LP problems. This method was effective in discarding invalid and sub-optimal solutions[7]. Moreover, CoPhy added a significant improvement by guaranteeing the utilization of a unique index option for each generated query plan and ensuring the indexes stayed within the given storage budget. However, solving large problems with LP solvers is challenging, often resulting in sub-optimal solutions[7].

Transition towards Modern Open-source Approach

The more recent advancements in index selection led to the development of open-source indexing tools, such as Dexter. Dexter was introduced in 2017 and adopted a two-phased approach, first generating a query plan and then grouping queries based on their template but with differing parameter values. It then created virtual indexes based on indexing plans, requested cost estimations from the query optimizer, and finally selected the index with the lowest query plans[10]. However, Dexter with limitations that do not consider deleting existing indexes and can not contain more than two indexes, restricting its flexibility[10].

Trend of Machine Learning Approach

The emergence of machine learning methods have marked a significant turning point in auto-indexing problem solutions, as depicted in figure 4. This trend, initiated in 2017 by Kossmann, has been increasingly explored by researchers [10]. An innovative approach proposed in 2022 utilized Monte Carlo Tree Search (MCTS) for index tuning [21]. Rather than employing dynamic programming to calculate the state/action values in extensive state/action spaces—a process that becomes untenable due to significant computational time and memory demands, MCTS uses simulation-based techniques. It approximates the action value function `Q(s, a)` through sampled traces, following state transitions within the Markov Decision Process (MDP). The MCTS method doesn't need to estimate `Q(s, a)` for all state-action pairs, but rather conducts a search among the representative leaf nodes to find the optimal index solution.

Concurrently, Kraska et al. proposed an innovative index selection method that utilized statistical machine learning techniques, more specifically, multi-arm bandit (MAB) algorithms [11]. This method reformulates the indexing problem into a MAB problem, where an expert agent selects the next ”arm” (index) based on observations from a contextual feature space. The agent then observes the reward following the index selection, leading to a process where the expert agent continuously selects the index yielding the highest reward, guided by the tuning results.

Experiment Result Analysis

A meaningful comparison necessitates a standard set of evaluation metrics and experimental settings. Due to the different optimization goals inherent in the algorithms under consideration, we find it appropriate to use a mix of benchmarking standards.

For the heuristic and linear programming methods, Kossmann used the Transaction Processing Performance (TPC) standards as the benchmark [10], specifically TPC-H [1] and TPC-DS [2]. These benchmarks are designed to simulate large-scale transactional workloads with complex queries, providing a comprehensive measure of each algorithm's performance. TPC also applied for MCTS budget aware search[21]. We will evaluate these algorithms using metrics such as runtime, query cost, and relative workload cost.

On the other hand, machine learning approaches, such as the Multi-Arm Bandit (MAB) algorithms, require a different benchmark. Following Kraska et al. [11], they added the Hybrid Transactional and Analytical Processing (HTAP) benchmark on top of TPC standards. HTAP allows for real-time performance evaluation, providing immediate observations for the learning agents. This feature is particularly useful for these algorithms as it complements their dynamic learning mechanisms well.

Results and Comparison

The experimental results from various studies demonstrate the performance impact of different index selection strategies on TPC-H and TPC-DS workloads.

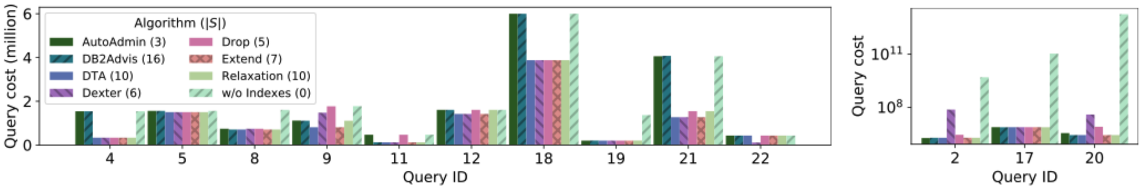


Figure 5: Estimated query processing costs for TPC-H (SF 10) on PostgreSQL with budget 5 GB[10]

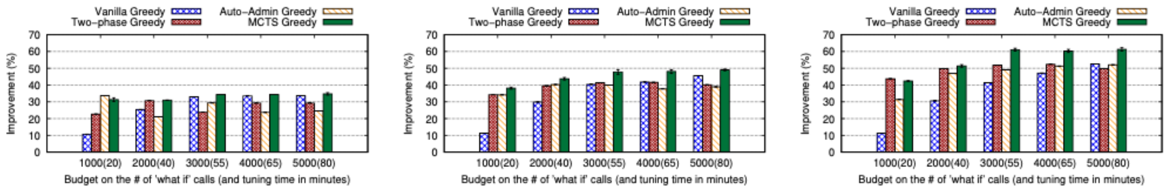
Figure 5 presents results from Kossmann et al.[10] on TPC-H benchmarks, emphasizing the considerable performance improvement through indexing compared to any algorithms, particularly for complex queries. Heuristic approaches, such as DROP and DTA, consistently outperform other methods in terms of query cost, most notably in queries 4, 18, and 21. However, this is not to say that LP-approach and machine learning are not competitive. These methods exhibit strong performance in queries 5 and 22.

Algorithm	Configurations	Index simulations	Cost requests			Runtime		
			Total	Non-cached	Cache rate	Total	Simulation	Costing
AutoAdmin	129	10 991	33 851	11 676	65.5%	2.1m	2.0%	95.9%
Naive-2	816	73 504	240 441	73 440	69.4%	15.3m	2.0%	66.5%
CoPhy	3 983	3 982	394 317	52 177	86.8%	10.1m	0.6%	94.9%
DB2Advis	2	7 179	180	180	0.0%	0.1m	24.0%	58.7%
DTA	1 442	25 812	1 650 510	129 811	92.1%	32.2m	0.4%	87.2%
Dexter	2	3 982	180	180	0.0%	0.4m	n/a	n/a
Drop	203	29 144	2 601 450	18 348	99.3%	35.0m	0.6%	19.7%
Extend	594	11 295	812 430	53 472	93.4%	12.8m	0.5%	84.1%
Relaxation	1 898	51 680	2 982 690	170 863	94.3%	60.7m	0.4%	66.6%

Figure 6: Algorithm cost comparison for TPC-DS (SF 10), storage consumption≈5 GB. Index simulations refer to the number of non-unique created hypothetical indexes, adapted from Kossmann et al.[10]

Similarly, Figure 6 shows that heuristic methods lead to a high cache rate (above 90%) with lower runtime costs in TPC-DS benchmarks. LP-methods constrain the budget, leading to a lower cache rate. However, it's important to note that these performances highly depend on the configuration settings. For instance, under a 5 GB budget, CoPhy only reduces the cost of requests by approximately 5%, possibly because the LP solver requires more time to solve complex problems.

Kraska and Zhang [21] illustrated how machine learning approaches significantly improve performance compared to queries without DBA (indexing), as shown in Figure 7 and Figure 8. The advantage of

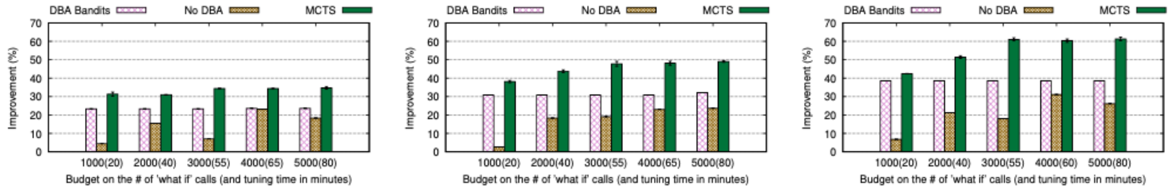


(a) $K = 5$

(b) $K = 10$

(c) $K = 20$

Figure 7: End-to-end performance comparison on TPC-DS with budget-aware greedy approach [21]



(a) $K = 5$

(b) $K = 10$

(c) $K = 20$

Figure 8: End-to-end performance comparison on TPC-DS with existing RL approaches[21]

MCTS becomes more pronounced as the scale factor increases, outperforming traditional algorithms like Greedy, AutoAdmin, and Two-phase approaches.

Workload	Tool	Recommendation	Creation	Execution	Total
TPC-H (Static)	PDTool	0.6	2.45	46.35	49.4
TPC-H (Static)	MAB	0.08	5.66	55.64	61.38
TPC-DS (Static)	PDTool	44.86	1.45	302.63	348.94
TPC-DS (Static)	MAB	1.53	5.94	242.15	249.62
TPC-H (Dynamic)	PDTool	1.55	9.36	26.35	37.25
TPC-H (Dynamic)	MAB	0.12	9.74	25.14	35
TPC-DS (Dynamic)	PDTool	11.13	6.08	187.08	204.29
TPC-DS (Dynamic)	MAB	1.66	16.48	155.65	173.79
TPC-H (Random)	PDTool	7.55	14.68	84.14	106.37
TPC-H (Random)	MAB	0.08	7.06	80.43	87.57
TPC-DS (Random)	PDTool	310.22	8.23	323.57	642.01
TPC-DS (Random)	MAB	1.4	19.81	227.02	248.24

Table 2: Total time breakdown for analytical workloads between PDTools and MAB (in minutes)[11]

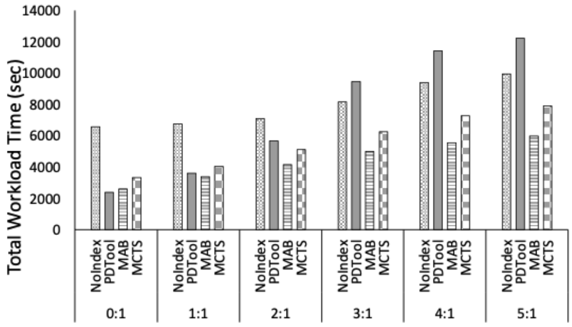


Figure 9: Transaction Analytic Ratio between MAB and MCTS budget-aware approach[21]

MAB, proposed by Kraska et al.[11], also demonstrates great performance across TPC-H and TPC-DS with various workloads, outperforming PDTools. This is especially true for recommendation tasks, which MAB accomplishes in a relatively low time cost, as illustrated in Table 2. When the situation becomes complex, the superiority of MAB and MCTS over PDTools and other linear programming methods becomes more evident, as shown in Figure 9.

Discussion: Evolutionary Path of Index Selection

The evolution of index selection in self-driving databases signifies a critical journey from heuristic techniques to linear programming, and ultimately, machine learning approaches. This progression has witnessed substantial improvements in the operational efficiency of databases, primarily due to the advancement in optimization methods.

In the initial stages, heuristic approaches like the DROP algorithm and AutoAdmin laid the foundation for index selection, facilitating efficient elimination of irrelevant indexes and harnessing constraints on feature spaces[19, 3]. However, these methods demonstrated certain limitations, including a heavy reliance on data attributes and a lack of guarantee for an optimal solution, particularly in the case of multi-column indexes.

The transition to linear programming methods was marked by the introduction of DB2 Advisor and CoPhy[18, 7]. These methods contributed to refining the performance through optimizer cost estimation and addressing the index optimization problem by formulating it as an LP problem. Nevertheless, these approaches sometimes yielded sub-optimal solutions and encountered difficulties in handling large problem sizes.

Recent advancements in machine learning techniques, specifically reinforcement learning and deep reinforcement learning, have drastically transformed the landscape of index selection. The utilization of Monte Carlo Tree Search (MCTS) for index tuning and the multi-arm bandit (MAB) algorithms have provided promising solutions to minimize costs on indexes storage and operation respectively[21, 11].

The performance of these index selection techniques has been tested on two widely-used benchmarking platforms: TPC-H and TPC-DS. While the performance results demonstrated significant improvements, some limitations were identified due to the complexity of these platforms and the varying definition of budget constraints. Moreover, unknown benchmarking settings and versions further complicate the performance evaluation.

Based on these improvements, we can expect that the application of more advanced machine learning methods in index selection will continue to drive the evolution of self-driving databases. These techniques offer promising solutions to improve the overall performance and efficiency of the system, in terms of cost of indexes storage and operation.

Challenges and Limitations on Index Selection Algorithms

Index selection in self-driving databases is a complex process, presenting several challenges:

- **Index Identification:** Extracting pertinent indexes from extensive workload data is a computationally intensive process demanding considerable time and resources.

- **Evaluation Complexity:** Assessing the performance of distinct index selection algorithms can be intricate due to the variety in optimization goals and the broad definitions of budget constraints.
- **Understanding Query Plans:** The task of index selection necessitates an in-depth comprehension of query plans, which are often complex and dynamic.

With respect to benchmarking, although TPC-H and TPC-DS are commonly employed, their complexity can pose its own challenges:

- **Unknown Benchmark Settings:** The undisclosed settings and versions used in benchmarking can influence the evaluation of the performance of index selection algorithms, thereby affecting their comparative study.
- **Variety in Benchmarking Platforms:** The usage of diverse benchmarking platforms by different researchers can further complicate the horizontal comparison of the various algorithms.

In spite of these challenges, continuous research in this field, particularly the implementation of machine learning techniques, promises significant advancements in enhancing the efficiency and performance of self-driving databases. Addressing these challenges effectively is vital for unlocking the full potential of these evolving techniques in the landscape of database management.

Conclusion

The evolution of index selection algorithms from heuristic methods to linear programming and the current trend of machine learning applications has significantly changed the landscape of self-driving databases. The advances in these algorithms as demonstrated in TPC-H and TPC-DS have brought forth substantial improvements in improving the quality of index selection, lowering index storage and operational cost like execution time. However, the challenges and limitations encountered during the index selection process emphasize the importance of further research domain. Machine learning techniques in index selection reveal a promising trajectory toward enhancing the overall efficiency of self-driving databases and continuing research on this track driving databases autonomously.

Future Work

Index selection in self-driving databases with the adoption of machine learning techniques making an exciting milestone. These advancements, particularly seen in the MCTS and MAB algorithms indicate the potential for machine learning to optimize storage and operational cost by selecting optimal indexes. Future studies can deepen this exploration, experimenting with new machine learning techniques or utilizing deep learning approaches to lower the operational cost in action planning.

Future work can also focus on improving evaluation metrics for index selection algorithms through standardized benchmarking. Exploration of multi-object optimization can be beneficial given that index selection differs in optimization goals across application platforms and scenarios. Lastly, define a common standard 'budget' parameter in the optimization process to lead to a holistic understanding of index selection algorithms' performance.

Group Reflection

how team works
- no team leader, everyone proposed ideas and agreed with the majority votes
- regular meetings throughout the documentation
- rapid update using network
- after determining the main topic, divide the team into two groups, two responsible for indexing selection (Runqiu, Sunchuangyu), and two (Xiaoyi, Qingxuan) for query optimization. Work separately but keep communicating about the main concept and ideas.
- finalizing report as a team including abstraction and conclusion section.

Individual Reflection

Xiaoyi Liu

Qingxuan Yang

Runqiu Fei

Sunchuangyu Huang

Responsibility:
contribution: indexing selection

References

[1] Transaction Processing Performance Council (TPC). *TPC Benchmark H Standard Specification*. Technical Report. 2023. URL: https://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.17.1.pdf.

[2] Transaction Processing Performance Council (TPC). *TPC-DS Benchmark Standard Specification*. Technical Report. 2023. URL: https://www.tpc.org/tpc_documents_current_versions/pdf/tpc-ds_v2.1.0.pdf.

[3] Sanjay Agrawal, Surajit Chaudhuri, and Vivek R. Narasayya. "Automated Selection of Materialized Views and Indexes in SQL Databases". In: *VLDB '00: Proceedings of the 26th International Conference on Very Large Data Bases*. 2000, pp. 496–505.

[4] Mumtaz Ahmad et al. "Predicting completion times of batch query workloads using interaction-aware models and simulation". In: *Proceedings of the 14th International Conference on Extending Database Technology*. 2011, pp. 449–460.

[5] Surajit Chaudhuri and Vivek Narasayya. "Anytime Algorithm of Database Tuning Advisor for Microsoft SQL Server". June 2020. URL: https://www.microsoft.com/en-us/research/publication/anytime-algorithm-of-database-tuning-advisor-for-microsoft-sql-server/.

[6] Surajit Chaudhuri and Vivek R Narasayya. "An efficient, cost-driven index selection tool for Microsoft SQL server". In: *VLDB*. Vol. 97. San Francisco. 1997, pp. 146–155.

[7] Debabrata Dash, Neoklis Polyzotis, and Anastasia Ailamaki. "CoPhy: A Scalable, Portable, and Interactive Index Advisor for Large Workloads". In: *Proc. VLDB Endow.* 4.6 (Mar. 2011), pp. 362–372. DOI: 10.14778/1978665.1978668.

[8] Jennie Duggan et al. "Performance prediction for concurrent database workloads". In: *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. 2011, pp. 337–348.

[9] Jan Kossmann and Rainer Schlosser. "Self-driving database systems: a conceptual approach". In: *Distributed and Parallel Databases* 38 (2020), pp. 795–817.

[10] Jan Kossmann et al. "Magic Mirror in My Hand, Which is the Best in the Land? An Experimental Evaluation of Index Selection Algorithms". In: *Proc. VLDB Endow.* 13.12 (July 2020), pp. 2382–2395. DOI: 10.14778/3407790.3407832.

[11] Tim Kraska et al. "No DBA? No regret! Multi-armed bandits for index tuning of analytical and HTAP workloads with provable guarantees". In: *Proceedings of the 2022 International Conference on Management of Data*. 2022.

[12] Lin Ma. "Self-Driving Database Management Systems: Forecasting, Modeling, and Planning". PhD thesis. Carnegie Mellon University, 2021.

[13] Lin Ma et al. "MB2: decomposed behavior modeling for self-driving database management systems". In: *Proceedings of the 2021 International Conference on Management of Data*. 2021, pp. 1248–1261.

[14] Barzan Mozafari et al. “Performance and resource modeling in highly-concurrent OLTP workloads”. In: *Proceedings of the 2013 acm sigmod international conference on management of data*. 2013, pp. 301–312.

[15] Dushyanth Narayanan, Eno Thereska, and Anastassia Ailamaki. “Continuous resource monitoring for self-predicting DBMS”. In: *13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*. IEEE. 2005, pp. 239–248.

[16] Andrew Pavlo et al. “Make your database system dream of electric sheep: towards self-driving operation”. In: *Proceedings of the VLDB Endowment* 14.12 (2021), pp. 3211–3221.

[17] Andrew Pavlo et al. “Self-Driving Database Management Systems.” In: *CIDR*. Vol. 4. 2017, p. 1.

[18] Gary Valentin et al. “DB2 advisor: An optimizer smart enough to recommend its own indexes”. In: *Proceedings of 16th International Conference on Data Engineering (Cat. No. 00CB37073)*. IEEE. 2000, pp. 101–110.

[19] Kyu-Young Whang. “Index selection in relational databases”. In: *Foundations of Data Organization*. Springer, 1987, pp. 487–500.

[20] Dong Young Yoon, Ning Niu, and Barzan Mozafari. “Dbsherlock: A performance diagnostic tool for transactional databases”. In: *Proceedings of the 2016 international conference on management of data*. 2016, pp. 1599–1614.

[21] Yuhao Zhang and Tim Kraska. “Budget-aware index tuning with reinforcement learning”. In: *Proceedings of the 2022 International Conference on Management of Data*. 2022, pp. 1528–1541.