SCIENCE CHINA
Information Sciences

• REVIEW •

February 2023, Vol. 66 121102:1–121102:23
https://doi.org/10.1007/s11432-021-3578-6

Survey on performance optimization for database systems

Shiyue HUANG[1], Yanzhao QIN[1], Xinyi ZHANG[1], Yaofeng TU[2], Zhongliang LI[2] & Bin CUI[1,3]*

[1]*Key Laboratory of High Confidence Software Technologies (MOE), School of Computer Science, Peking University, Beijing* 100871, *China;*
[2]*ZTE Corporation, Nanjing* 210012, *China;*
[3]*Institute of Computational Social Science, Peking University (Qingdao), Qingdao* 266555, *China*

Received 20 December 2021/Revised 8 March 2022/Accepted 11 July 2022/Published online 11 January 2023

Abstract The performance optimization of database systems has been widely studied for years. From the perspective of the operation and maintenance personnel, it mainly includes three topics: prediction, diagnosis, and tuning. The prediction of future performance can guide the adjustment of configurations and resources. The diagnosis of anomalies can determine the root cause of performance regression. Tuning operations improve performance by adjusting influencing factors, e.g., knobs, indexes, views, resources, and structured query language (SQL) design. In this review, we focus on the performance optimization of database systems and review notable research work on the topics of prediction, diagnosis, and tuning. For prediction, we summarize the techniques, strengths, and limitations of several proposed systems for single and concurrent queries. For diagnosis, we categorize the techniques by the input data, i.e., monitoring metrics, logs, or time metrics, and analyze their abilities. For tuning, we focus on the approaches commonly adopted by the operation and maintenance personnel, i.e., knob tuning, index selection, view materialization, elastic resource, storage management, and SQL antipattern detection. Finally, we discuss some challenges and future work.

Keywords database management system, performance optimization, performance prediction, anomaly diagnosis, database tuning

Citation Huang S Y, Qin Y Z, Zhang X Y, et al. Survey on performance optimization for database systems. Sci China Inf Sci, 2023, 66(2): 121102, https://doi.org/10.1007/s11432-021-3578-6

1 Introduction

Database performance optimization is critical from the perspectives of users and the operation and maintenance personnel of database systems. With the ever-increasing complexity and variety of database workloads, database application systems have imposed significant challenges on performance requirements [1]. It is common to restrict the percentage of queries or transactions with bad performance in service-level agreements (SLAs). Thus, a significant job of the operation and maintenance personnel is a performance optimization, i.e., to monitor the running states of the database system and solve the problems of performance regressions.

We start from the measurement and influencing factors of database performance and summarize three topics related to optimization from the perspective of the operation and maintenance personnel, i.e., performance prediction, anomaly diagnosis, and tuning.

1.1 Performance measurement

Typically, database performance is measured by the throughput and latency of transactions or queries. In some scenarios, resource consumption is also considered a part of performance metrics. For discussion in this review, we only take throughput and latency as measurements.

*Corresponding author (email: bin.cui@pku.edu.cn)

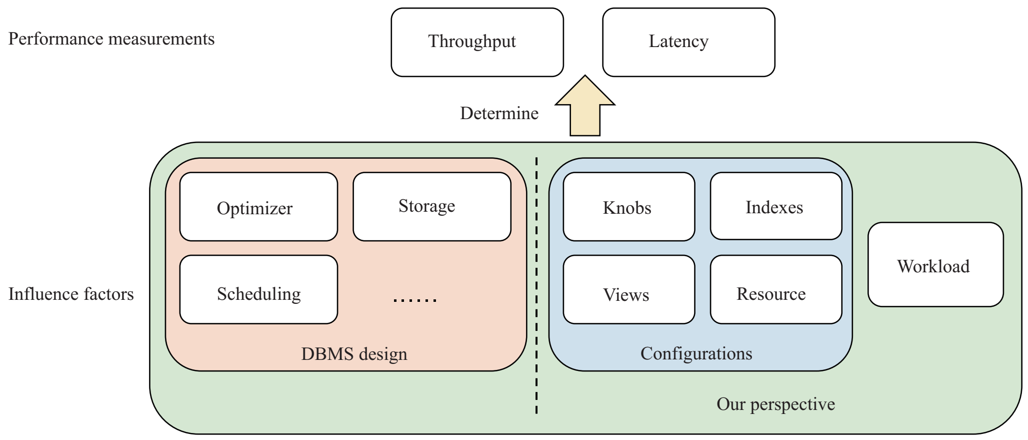

Figure 1 (Color online) Influencing factors of database performance.

Throughput refers to the number of transactions or queries processed in a certain period of time, e.g., the number of transactions per second. It is determined by the arrival rate of workload queries and the processing ability of the database system. When a resource bottleneck is reached, the throughput will not increase even if the workload rate grows high. The expectation from database users is to maximize the throughput with a certain workload pattern.

Latency refers to the response time of a transaction or query. It consists of the waiting time for resources and the processing time of queries. There are two expectations on latency from database users, which are also common in SLAs. First, the latency of most queries should be low. Second, the latency for slow queries should be limited.

1.2 Influencing factors

All performance optimization techniques are based on the influencing factors of database performance. As shown in Figure 1, the database management system (DBMS) design, configuration, and workload can influence the performance of a database system. The designs of the optimizer [2], storage layer, scheduling [3], and other components of the DBMS determine its ability to provide data services. These issues are mostly considered by developers of database systems. In this review, we only focus on the configuration and workload because the configuration is tunable, and the workload could be predictable from the perspective of the operation and maintenance personnel. Detailed factors are shown as follows.

Knobs. Knobs can decide how a database system works, such as the buffer pool size, logging level, and autovacuum strategy. Changes on different knobs may influence the performance in different ways.

Indexes. Indexes can be built on single or multiple columns, so the number of potential indexes is large. They may speed up scans and projections but slow down data modifications.

View materialization. The intermediate results of queries can be stored and reused, and this process is called view materialization. Although this technique can speed up some queries, the storage overhead should be limited.

Hardware and resource limitation. The maximum throughput is limited by the hardware and resource. When a bottleneck is reached, some transactions should be blocked to wait for the resource, and the latency should increase. Moreover, hardware anomalies may result in intermittent slow queries or failures, which may also influence the overall performance.

Workload. On the one hand, the queries, arrival rates, and concurrency of workload transactions directly determine the database performance. Based on the workload patterns, the performance can be predicted as guidance for optimization. On the other hand, poor design of workload queries can lead to poor performance. For example, functions with large scans in structured query language (SQL) queries may slow down transactions, certain execution sequences can easily trigger lock waits, and frequent deletions may cause vacuum problems. Hence, it is meaningful to detect such queries and rewrite them to improve database performance.

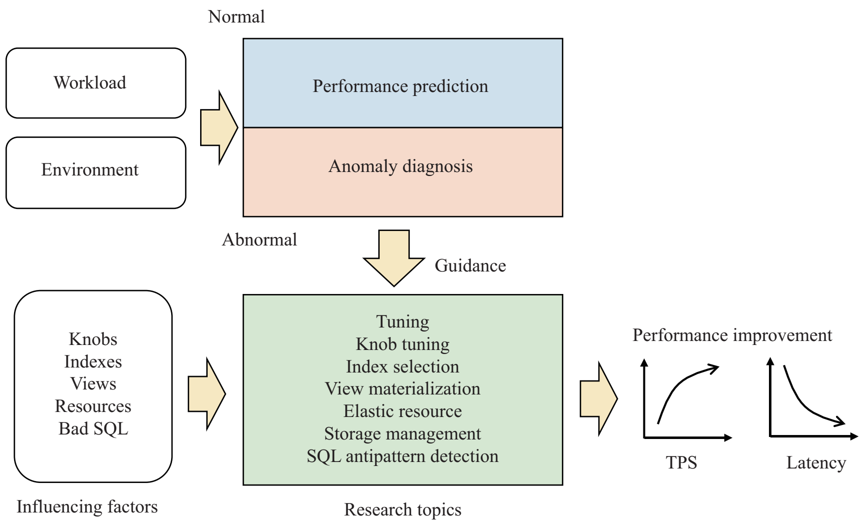

Figure 2 (Color online) Relation between influencing factors and research topics.

1.3 Research topics

Database performance optimization aims at improving the database throughput and latency through different approaches, which are consistent with the influencing factors of the database performance, as shown in Figure 2. In normal situations with certain workload patterns, the prediction of performance may provide some guidance for the adjustment of configurations and resources. When anomalies occur in the workload or environment, the diagnosis system should determine the root cause, which helps database administrators (DBAs) solve the problem. Both situations require certain tuning actions on the configurations or SQLs so that improvements can be achieved.

Consistent with this procedure, research topics on database performance are mainly categorized as follows.

Performance prediction. Given a certain database environment and workload pattern, the throughput and latency of queries may not vary too much due to the stability of common database systems. Some researchers focus on how to predict this information as guidance for optimization ahead of time.

Anomaly diagnosis. The root cause of performance anomalies could be discovered by analyzing monitoring data and logs. To improve the efficiency of this procedure, several diagnosis algorithms and tools have been developed by researchers as a part of the operation and maintenance system.

Tuning. Tuning refers to the adjustment of the influencing factors of database performance. For safety and stability, modifying the DBMS to improve the performance is not recommended. Hence, we only focus on the influencing factors that are tunable, i.e., knobs, indexes, views, resources, and SQL antipatterns. Automatic optimization of these factors has been studied for decades, and some AI-based techniques have been proposed in recent years.

Database performance optimization has been researched for years. Traditional techniques basically rely on heuristic rules, such as thresholds and greedy algorithms. Recently, with the development of artificial intelligence (AI), machine learning models have been widely used to benefit database systems. A number of algorithms and systems have been proposed to employ machine learning models for database performance optimization. Zhou et al. [4] surveyed the research on the interaction of databases and AI, which includes the learning-based techniques of database configuration and monitoring for performance optimization. Nevertheless, database performance optimization is still a challenging task for complex, dynamic real-life database environments.

1.4 Contributions

This review makes the following contributions.

(1) We review some typical research work on database performance prediction.

(2) We review the database performance anomaly diagnosis techniques, which are categorized by their input data, including monitoring metrics, logs, and time metrics.

(3) We review the typical database tuning techniques, including knob tuning, index selection, view materialization, elastic resource, storage management, and SQL antipattern detection.

(4) We address research challenges and future directions for database performance issues.

We organize the rest of the review as follows: Section 2 provides an overview of datasets related to experiments on database performance optimization. Sections 3–5 review research work on database performance prediction, anomaly diagnosis, and tuning, respectively. Section 6 provides some challenges in the related field and future research directions. Section 7 shows a summary of this review.

2 Related datasets

Before reviewing specific research work, we give an overview of the related datasets in this field. Most of the research work on database performance optimization takes open-source benchmarks as the datasets for their experiments. These benchmarks contain both the data and the workload, as a simulation of a real-world database system.

In the experiments, researchers create the databases on the target system and run the workloads. For performance prediction tasks, they collect the necessary data, such as the monitoring metrics, throughput, and latency, during the execution of the workload. Then they test the effectiveness of their proposed methods with the collected data. For performance anomaly diagnosis tasks, they manually inject some anomalies into the workload, such as bad SQLs and resource pressure. Then they collect the necessary data during execution and test their methods. For tuning tasks, they either continuously apply the tuning progress throughout the workload, or tune the database at the beginning and check the effects.

The workloads of the benchmarks are generally classified into online transaction processing (OLTP) and online analytical processing (OLAP), i.e., online transaction processing and online analytical processing.

Typical OLTP benchmarks are listed as follows.

TPC-C1). A widely-used benchmark of order processing. It involves 5 transaction types on 9 tables with the number of warehouse entries as a scaling factor.

TPC-E2). A benchmark of the activities of a stock brokerage firm. It is more complex than TPC-C, involving 12 transaction types on 33 tables.

YCSB [5]. Yahoo! Cloud Serving Benchmark, a benchmark for cloud data serving systems. It involves 6 transactions on 1 table.

TATP3). A benchmark that simulates a typical home location register (HLR) database. It involves 7 transaction types on 4 tables.

Smallbank [6]. A benchmark on a small banking database. It involves 6 transaction types on 3 main tables.

Typical OLAP benchmarks are listed as follows.

TPC-H4). A popular benchmark with business-oriented queries on 8 tables.

TPC-DS5). A decision support benchmark that models a retail warehouse. It involves 7 fact tables and 17 dimension tables.

JOB [7]. Join Order Benchmark, a workload for testing the performance of join operations. It is based on the IMDB dataset6), which involves 22 tables.

Some research work directly uses the data collected from real-life databases instead of the benchmarks. For example, ISQUAD [8] uses the monitoring data of the service workloads on Alibaba OLTP database, where the anomalies of intermittent slow queries are manually labeled by the DBAs. Such datasets are not easily available, because they may involve the privacy of the database users, and the data collection and labeling are expensive. However, they are more convincing as an application in real-life situations.

1) http://www.tpc.org/tpcc/.
2) http://www.tpc.org/tpce/.
3) http://tatpbenchmark.sourceforge.net/.
4) http://www.tpc.org/tpch/.
5) http://www.tpc.org/tpcds/.
6) http://homepages.cwi.nl/ boncz/job/imdb.tgz.

Table 1 Summary of research on database performance prediction

Prediciton target	Research work	Technique
Single query	Ganapathi et al. [12]	KCCA for multiple metrics
	Akdere et al. [13]	Plan-level & oprator-level modeling
	Wu et al. [14]	Calibrate the optimizer cost model
	Marcus et al. [15]	Plan-structured DNN
Multiple queries	DBSeer [9]	White-box & black-box models
	Zhou et al. [16]	Graph neural network
	ModelBot2 [11]	OU-models & interference model

3 Performance prediction

Database performance prediction aims at estimating the execution time (i.e., latency) of queries or the throughput for given environments. It may function as part of a monitoring system such as DBSeer [9], or a component of a self-driving database system [10] such as ModelBot2 [11] for NoisePage.

The input of a performance prediction system should contain sufficient information about the situation, i.e., the workload and the environment settings. The workload can be described as arrival rates of each type of transactions, or with more detailed information like the query plans. The environment settings may include resource limitations, configuration knobs, and hardware information. If it is assumed that some settings would not change during the application, and would not influence the estimated result, such information can be omitted.

The research work on performance prediction can be categorized by the prediction targets, i.e., single query and multiple queries, as shown in Table 1 [9,11–16]. We do not discuss the work on cost estimation of the optimizer, because such cost is designed for query plan generation and scheduling, instead of indicating the execution time. Detailed reviews are as follows.

3.1 Single query

Single query performance prediction takes the features of single queries as the input of their models. A common limitation is that they do not consider the influence of the concurrent execution of other queries.

Ganapathi et al. [12] proposed to use kernel canonical correlation analysis (KCCA) [17] to predict multiple metrics of a single query, i.e., elapsed time, disk I/O, message count, message bytes, records accessed, and records used. The first step is to generate a query plan feature vector, which consists of the instance count and cardinality sum of each possible operator. The second step is to infer its coordinates on the performance projection using its k-nearest neighbors, where the distance is computed with the Gaussian kernel [18]. Finally, the performance projection is mapped back to the metrics.

This technique is applicable when a different database with a different schema is accessed, because of the property of KCCA. It supports both short and long-running queries, but KCCA is not fast enough for the practical use of short-running queries. The prediction accuracy is also limited by the quality of the training data of KCCA; i.e., similar queries in the training set are necessary.

Akdere et al. [13] proposed plan-level and operator-level machine learning techniques for single query performance prediction. The plan-level technique employs one model for all query plans. The input features of the model consist of query optimizer estimates, such as estimated plan total cost, and the counts/rows of each type of operator. The operator-level technique is more fine-grained. It employs several models, each for a single type of operator. The input features of a model consist of both the estimates on the operator and the start/run time of its children operators. Based on these techniques, a hybrid modeling technique is proposed, which uses the plan-level model for materialization sub-plans and operator-level models for others, which avoids the high error of the operator-level models on materialization sub-plans.

Empirical studies show that hybrid modeling is feasible and effective for both static and dynamic workloads. It assumes the independence of different operators in operator-level modeling, which still remains to be discussed [11].

Wu et al. [14] proposed to calibrate the cost estimation of the optimizer to predict query execution time. They refine both the cost units and the cardinality estimation. For cost units, they design some particular calibration queries that isolate each unit parameter from others. For cardinality estimation, they propose

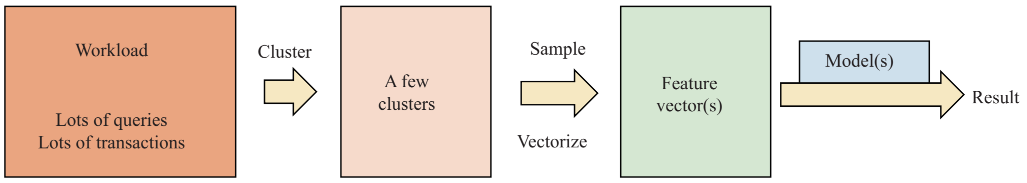

Figure 3 (Color online) Prediction procedure for multiple queries.

a sampling-based technique that increases the overhead but achieves better quality. Since the refinement procedure is performed once per query, rather than once per plan, such overhead is considered affordable.

The idea of calibrating the cost estimation is novel, while the prediction accuracy still relies on the effectiveness of the optimizer cost model.

Marcus et al. [15] proposed to use a plan-structured deep neural network (DNN) to predict the latency of a single query. The forward propagation of DNN follows the tree structure of the query plan. There is either a leaf neural unit or an internal neural unit for each operator. The leaf neural unit takes the features of the operator as input, and outputs both the estimated latency and a data vector. The internal neural unit takes the output of children's neural units and the features of the operator as input, and also outputs the estimated latency and a data vector. The loss function is defined as the error of latency estimation.

Similar techniques are also applied to cardinality and cost estimation in other research work [19]. The prediction accuracy strongly relies on the size and quality of the training data.

3.2 Multiple queries

Multiple queries performance prediction considers the influence of concurrency. The basic procedure is shown in Figure 3. First, they take the whole workload as input, and cluster the transactions or queries to generate a small number of significant clusters. Then, they transfer the sampled queries of the clusters into vectors, as the input of one or several models to generate the result. They either calibrate the single query performance prediction technique with concurrency modeling, or build a thorough model for the whole workload.

DBSeer [9] is a tool for database monitoring [20, 21]. It provides what-if APIs for database performance and resource prediction in highly-concurrent OLTP workloads. Given the ratio of each type of transaction and the number of transactions emitted per second (i.e., TPS), it predicts latency and resource consumption; or given the transaction ratios and the limitation on each resource, it predicts the maximum TPS.

DBSeer builds both white-box and black-box models for prediction. The system first clusters the transactions with DBSCAN algorithm [22], then uses the ratio of each cluster and total TPS to estimate the performance. The white-box models refer to iterative approaches and Monte-Carlo simulation. Specifically, based on the estimated distribution of data access and MySQL's adaptive flushing strategy, it constructs iterative formulas for the disk I/O of log-triggered data flushes. According to Thomasian's two-phase locking (2PL) [23], it constructs iterative formulas for the latency of lock contention. For the I/O of capacity misses, i.e., buffer pool misses, it performs Monte-Carlo simulation with an N-element list, where N is the number of pages of RAM. For the other resources, it employs black-box models, including both linear and non-linear (i.e., polynomial fitting, KCCA [17], decision trees, and neural networks) models. For example, it uses linear regression to estimate the consumption of the CPU.

A strong point of DBSeer is that it proposes several novel white-box models, which are more accurate than black-box models when the test data is far from training data for black-box. However, the white-box models are specific to the DBMS. For example, the model for log-triggered data flushes designed for MySQL may not work for a DBMS with a different strategy. Besides, ad-hoc transaction types are not supported. The types of transactions need to be known ahead of time, and there are pre-processing steps for the estimation of data access distributions. For a new transaction that the system has not seen before, the prediction could not work.

Zhou et al. [16] proposed to use graph embedding for performance prediction with concurrent queries. It takes the information of queries from the optimizer, e.g., query plans and estimated cardinality, and predicts the startup time, execution time, and resource consumption of memory and CPU.

Table 2 Summary of research on database performance anomaly diagnosis

Research work	Input	Output	Anomaly type
DBSherlock [25]	KPIs	Abnormal KPIs and anomaly types	Performance anomaly
ISQUAD [8]	KPIs	Abnormal KPIs and anomaly types	Intermittent slow query
FluxInfer [26]	KPIs	Root cause KPIs	Performance anomaly
Dundjerski et al. [28]	KPIs	Anomaly Types	General Anomaly
Distalyzer [29]	Debug log	Differences in event and state features	Abnormal system behavior
Sentinel [30]	Debug log	Differences in events and transitions	Abnormal system behavior
ADDM [31]	DB time	Bottleneck resource	Performance bottleneck
ProtoXplore [32]	RATP	Attributions of contentions	Resource contention

The graph represents the relationships within a set of queries. Each vertex represents an operator in a query plan. The vertex features consist of estimated cost, operator type, predicates, and sample bitmap. Each edge represents the relation between the operators, including parent-child, data sharing, data conflict, and resource competition. The weight of an edge is the weighted sum of the relations multiplied by the time overlap factor. After building the graph, a three-layer graph neural network [24] is employed, which outputs an embedding vector for each vertex. Finally, a graph prediction network takes the embedding as input, and estimates the performance matrix.

The strong points of this work include that, it proposes an accurate graph embedding model for concurrent queries, and supports dynamic workload by graph updates. However, there are still some limitations. First, it is difficult to build a graph for a complex transaction mixture. The edge weights rely on weights of each relationship and time overlap factor computed from model outputs, which is complex for highly-concurrent workloads. Second, it does not consider pipelined query execution, changed plans, knobs modification, and transfer of hardware environment. Finally, it relies on cost estimation from the optimizer, which could be inaccurate [7, 14] and affects the result of prediction.

ModelBot2 [11] (MB2) works as part of a self-driving database system [10]. It extracts operating units (OUs) from queries and predicts their time and resource consumption of them.

The OUs are independent operations specific to a DBMS, which should be comprehensive and have low-dimensional features. MB2 uses an OU-model to predict time and resources for each single OU, which takes the OU features, behavior knobs, and hardware context as input. Then, it uses an interference model to predict the influence of concurrency, and the model is shared by all OUs. The input of the interference model consists of the output of an OU-model and the summary statistics of other concurrent OUs, i.e., the sum and variance of time and resources. The output is the ratio between the actual metrics and the OU-model's output.

There are several strong points of MB2. First, it proposes an independent, low-dimensional, comprehensive OUs design. Second, it handles concurrency with an accurate interference model. Finally, empirical studies show that MB2 has good generalization, adaptation, and robustness. The limitations include that, it requires manually designed OUs and features, and it is designed only for in-memory DBMS. Also, it relies on cardinality estimation from the optimizer, which could be inaccurate in some conditions.

4 Anomaly diagnosis

Performance anomalies can be discovered through the drop of throughput or the long latency of certain queries. The target of anomaly diagnosis is to identify the root causes, which vary in different research work. DBSherlock [25] and FluxInfer [26] identify anomalous key performance indicators (KPIs) [27]. DBSherlock can also identify anomaly types, and so do ISQUAD [8] and Dundjerski et al. [28]. Distalyzer [29] and Sentinel [30] focus on system behaviors. ADDM [31] and ProtoXplore [32] only analyze resource issues. Table 2 [8, 25, 26, 28–32] summarizes the data sources and abilities of the proposed systems in these studies.

Based on the input data, we categorize these techniques as follows.

4.1 Monitoring metrics based

DBSherlock, ISQUAD, FluxInfer, and Dundjerski et al. take the system and database KPIs, i.e., monitoring metrics [33] as input, including both resource and workload monitoring. Resource metrics, i.e., CPU, memory, I/O, and network, are gathered from system states. Workload metrics, e.g., the number of rows read, are acquired from certain database views and logs.

DBSherlock [25]. DBSherlock is diagnosis module of the monitoring tool DBSeer. It allows a user to specify the normal and anomalous time regions, and finds predicate anomalies with heuristic rules. Then it incorporates domain knowledge and causal models to generate explanations. The output includes the anomalous predicates, as well as the explanations if causal models are specified. It also supports automatic anomaly detection with DBSCAN algorithm.

DBSherlock supports highly concurrent and complex OLTP workloads, as well as general performance anomalies. It allows incorporating domain knowledge, and supports automatic anomaly detection. It relies on user feedback for causal models, and invariants are not considered valid explanations, e.g. buffer pool size.

ISQUAD [8]. ISQUAD focuses on intermittent slow queries (iSQs), and provides explanations based on KPI patterns. First, it extracts KPI anomalies by identifying some patterns, i.e., spikes up or down, level shifts up or down, and even void. It applies Robust Threshold [34] for spikes and T-Test [35] for level shifts. Then, it employs confidence [36] based association rule learning for dependency cleansing, because KPIs are highly correlated with each other [37]. Afterward, it clusters the anomaly patterns with TOPIC algorithm, which is a modified KD-tree [38]. Finally, it extracts useful and suggestive information by using a Bayesian case model [39]. For online diagnosis, it finds the cluster of anomaly patterns whose similarity score to the instance is above a given threshold. If no cluster is found, the instance forms a new cluster, and its root cause needs to be labeled manually.

ISQUAD shows high accuracy in the diagnosis of iSQs with its novel algorithms. However, it is difficult to obtain enough labeled data for training. Also, a new cluster in the online stage has a cold start problem.

FluxInfer [26]. The target of FluxInfer is to distinguish between the root cause KPIs and symptom KPIs, instead of specifying the type of anomaly. It first detects anomalous KPIs with a robust anomaly detection algorithm, then constructs a weighted undirected dependency graph (WUDG) and uses PageRank [40] to find significant nodes as the root cause KPIs. The robust anomaly detection algorithm uses the Gaussian mixture model to cluster the KPIs for data smoothing, and measures the abnormality of the last change with the z-score. The WUDG takes KPIs as nodes and their dependencies as edges, where Fisher-Z Test [41] is used for the independence test. With the weighted PageRank algorithm on the WUDG, nodes with larger score ranks are considered root causes.

FluxInfer proposes WUDG to accurately represent the dependency relationships of KPIs, and the procedure can automatically localize the root-cause-related KPIs. It only gives KPI rankings instead of an explanation, which still requires further manual analysis.

Dundjerski et al. [28]. This review proposes an end-to-end automatic troubleshooting system on Azure SQL databases [42]. The system consists of 5 generic models, 11 categorization models, and a rule-based expert system. The generic models are designed for non-expected behavior, including query duration, timeout query, timeout DB, exceptions, and compile time. They compute some defined values based on telemetry monitoring data, and compare them with baselines. The categorization models aim to give potential explanations from specific patterns of monitoring data. The expert system makes the precise distinction by using IF-THEN statements. With these components, the system has overall coverage of different types of real case issues.

This system contains both data science models and an expert system. It has comprehensive anomaly types and good explainability. One problem is that all the models and rules are manually designed for certain a DBMS, which takes much human effort and could be difficult to transfer to another database environment.

4.2 Log comparison based

Distalyzer and Sentinel take two sets of logs as input, and output their major differences. If one is from a normal situation and the other is from an anomaly, the major differences of logs may indicate the root cause [43]. These techniques can make use of event information from logs, which is omitted by KPI-based techniques.

Distalyzer [29]. Distalyzer extracts event time features and state features from the logs. It detects the significant differences with Welch's t-test, and learns the dependencies with a dependency network [44]. Finally, it identifies the most notable results with a score function.

Sentinel [30]. Sentinel focuses on system behaviors, including event proportion, transition probability, and transition time. It uses the earth-mover's distance [45] between cumulative distribution functions to measure the difference in transition time. The system has a lower overhead than Distalyzer, because it gets log data from the memory instead of disk files.

4.3 Time analysis based

ADDM and ProtoXplore are based on the analysis of time metrics instead of all the monitoring metrics. These time-based diagnosis techniques use graphs for detailed analysis of certain anomalies, i.e., resource bottlenecks or contentions.

ADDM [31]. The automatic database diagnostic monitor (ADDM) for Oracle uses database time to diagnose performance bottlenecks. The database time is defined as the sum of the time spent inside the database processing user requests. It consists of the time for various phases to process the request, as well as the time to use and wait for resources. ADDM uses DBTime-graph to analyze the root cause of performance anomalies. The DBTime-graph is a directed acyclic graph to identify the categories through different dimensions of database time consumption. Different from a decision tree, ADDM explores all the children if the consumed database time is significant. It functions as a part of the self-managing framework of Oracle databases.

ProtoXplore [32]. ProtoXplore focuses on analyzing resource contention and showing hot resources, slow nodes, causing and receiving queries. It takes resource acquire time penalty (RATP) as the metric, which is defined as the per-unit time of waiting for and consuming a resource. It formulates the blame value for contention with blocked time computed from both full and partial overlap with concurrent tasks. It also builds a Proto-Graph for consolidating analysis on different granularities.

The Proto-Graph is a directed acyclic graph that consists of seven levels of nodes, which is also called iQC-Graph [46]. The nodes of the first two and last two levels represent the victim and concurrent queries and stages. Those of the middle three represent resource, host, and cause levels of contention. The edges represent the relations of data flow. The vertex contributions are computed from the RATPs. With the Proto-Graph, a user can analyze the critical path to blame attributions of different levels of contentions.

5 Tuning

Tuning techniques aim at increasing the throughput or reducing the latency for a given workload. There are also some techniques that try to reduce resource consumption with restrictions on throughput or latency. As mentioned in Subsection 1.2, the main influence factors of the database are configuration knobs, indexes, view materialization, hardware and resource allocation, and workload issues. Here we do not consider the design of the DBMS, which may not be modified for safety and stability. The corresponding research topics are as follows, with Table 3 [47–88] as a brief summary. The column 'ML' shows whether machine learning is involved, as learning-based techniques usually show good performance in modern applications [89].

5.1 Knob tuning

There are hundreds of configuration knobs in a database system. They affect the performance and resource consumption of the database. Given a target workload, database knob tuning aims to find a knob configuration that improves the database performance or reduces resource consumption. There are complex dependencies among knobs, and database knob tuning is prone to be an NP-hard problem [90]. So it is difficult to optimize the database performance by manually tuning the knobs. To this end, a number of automatic knob-tuning techniques have been proposed since the last decade. Based on the techniques used, they can be categorized into rule-based, Bayesian optimization (BO) based, and reinforcement learning (RL) based [91].

5.1.1 *Rule-based*

Rule-based methods recommend database configuration by hard-coded rules or heuristics. Database vendors develop tuning tools to provide DBA with knobs recommendations through determining the allocation of the DBMS's resources [47, 92, 93] or identifying database's bottlenecks due to misconfigurations [31]. In addition, BestConfig [48] tries to find a good configuration according to several heuristics.

Table 3 Summary of research on tuning techniques

Approach	Technique	ML	Research work
Knob tuning	Search based	No	STMM [47], BestConfig [48]
	BO based	Yes	iTuned [49], Ottertune [50], Tuneful [51], RelM [52], ResTune [53]
	RL based	Yes	CDBTune [54], Qtune [55]
Index selection	Pure benefit	No	Drop [56], AutoAdmin [57], Anytime [58], Dexter
	Per-storage benefit	No	DB2Advis [59], Relaxation [60], CoPhy [61], Extend [62]
	RL	Yes	Basu et al. [63], Sadri et al. [64, 65], Sharma et al. [66], Lan et al. [67]
	Comparison API	Yes	Ding et al. [68]
View materialization	Candidate generation	No	Dökeroglu et al. [69], DB2 UDB [70]
	Heuristics for selection	No	DB2 Design Advisor [70], BIGSUBS [71], CLOUDVIEWS [72]
	RL for selection	Yes	Yuan et al. [73], Liang et al. [74]
Elastic resource	Partition & placement	No	Accordion [75], E-Store [76], Clay [77], NashDB [78]
		Yes	P-Store [79]
	Migration	No	Albatross [80], Zephyr [81], Squall [82], MgCrab [83]
Storage management	Modified RDD	No	UlTraMan [84], Dragoon [85]
SQL AP detection	Antipattern detection	No	Shao et al. [86], Khumnin et al. [87], SQLCheck [88]
	SQL rewrite	No	Soar

Rule-based methods strongly depend on the assumptions of their heuristics. They do not evolve according to the interaction, thus, fail to utilize knowledge gained from previous tuning efforts. Most rule-based methods have restrictions on DBMSs since they are provided by database vendors and tailored to a particular DBMS. They tune specific knobs, and their resource modeling depends on the design of a DBMS.

STMM [47]. STMM is a rule based self-tuning memory manager released with DB2. It tunes both database memory heaps and cumulative database memory allocation. STMM tunes memory between radically different memory consumers such as compiled statement cache, sort, and buffer pools via a cost-benefit analysis technique.

BestConfig [48]. BestConfig searches the optimal knob configurations based on certainly given heuristics. It iteratively uses divide-and-diverge sampling heuristics. It divides the configuration space into sub-spaces represented by one sample and only keeps one sub-space according to the evaluation of the samples. Such a sampling procedure is time-consuming, and might trap in the local optimal.

5.1.2 *BO-based*

BO-based methods adopt BO to suggest promising knob configurations. They follow sequential model-based global optimization (SMBO) framework [94] to search for an optimal DBMS configuration: (1) fitting probabilistic surrogates to model the relationship between knob configurations and database performance and (2) choosing the next configuration to evaluate by maximizing acquisition function based on the surrogates. BO-based database knob tuning systems include iTuned [49], OtterTune [50], Tuneful [51], ReIM [52], and ResTune [53]

iTuned [49]. iTuned adopts Bayesian optimization to search for a well-performing configuration. It uses a stochastic sampling technique — Latin hypercube sampling [95] (LHS) for initialization. LHS attempts to distribute sample points evenly across all possible values over the configuration space. iTuned does not use the observations collected from previous tuning sessions to speed up the target tuning task.

OtterTune [50]. OtterTune presents a pipeline to conduct automatic tuning via machine learning-based techniques. (1) OtterTune selects the most impactful knobs by Lasso algorithms to prone the configuration space. (2) Then, it maps unseen database workloads to previous workloads to transfer experience. (3) Lastly, OtterTune recommends knob settings by maximizing acquisition function based on the surrogates fit on the observations from target and previous workloads. The workload mapping procedure maps the target workload based on the absolute measurements of internal metrics in DBMS, which might struggle to generalize across different hardware environments since the absolute measures of workloads diff a lot in the quantitative scales [53].

Tuneful [51]. Tuneful is designed for the analytics engine, such as Sparks [96]. It considers the dynamic characteristics of the analytics environment and proposes a tuning cost amortization model, which demonstrates that it is more beneficial to tune the recurrent workloads. Tuneful combines incremental sensitivity analysis based on Gini score and Bayesian optimization to identify near-optimal configurations from a high-dimensional search space, using a small number of executions. However, it follows the workload mapping strategy of OtterTune and still lacks an effective methodology to handle the dynamics of workloads.

RelM [52]. RelM studies the problem of automatic tuning the memory allocation for applications running on analytics engines. Relm constructs an empirically-driven "white-box" algorithm that models the impact of the memory configurations on the efficiency of the system resource utilization and the reliability of execution. It could provide a close-to-optimal tuning at a fraction of the overheads compared to state-of-the-art AI-driven "black box" algorithms. In addition, the "white-box" analytical algorithm is used to speed up Bayesian optimization. Relm highlights Limitations include that the "white-box" algorithm is customized for memory allocation in analytics systems, extending to other aspects or systems needs the redesign of domain experts.

ResTune [53]. ResTune is designed to optimize the resource utilization of database systems without violating the SLA constraints on the throughput and latency requirements. It adopts a constrained Bayesian optimization technique that extends Bayesian optimization with an inequality-constrained setting [97]. To further speed up the target tuning task, it proposes an ensemble framework that combines base-learners of previous workloads and generates a meta-learner for the target workload. Unlike previous studies, the ensemble framework uses relative rankings rather than absolute distances to measure the similarity between previous and the target workloads. It can better transfer knowledge across different hardware environments and achieve fast tuning and efficient adaptation. One potential issue of ResTune is that the setting of SLA constraints might be challenging when the running workload is non-static [98].

5.1.3 *RL-based*

RL-based methods adopt deep deterministic policy gradient (DDPG) [99], which is a policy-based model-free reinforcement learning agent. DDPG consists of two neural networks — actor and critic. The actor inputs internal metrics of the DBMS and outputs a promising action (i.e., configuration). The critic evaluates the selected configuration based on the reward. The RL-based methods model knob tuning as a Markov decision process (MDP) [100] and explore the configuration space via trial-and-error.

CDBTune [54]. CDBTune first adopts DDPG algorithm to tune the database's knobs. The superior representative ability of neural networks makes CDBTune well-performing in high-dimensional configuration space. However, the training of neural networks in DDPG needs lots of tuning samples, leading to high learning costs of CDBTune.

Qtune [55]. Qtune featurizes the query information in the given workload to support query-level tuning, while the previous studies only support coarse-grained tuning (i.e., workload-level tuning). Furthermore, Qtune clusters the queries based on their "best" knob values to support cluster-level tuning. In order to embed the query information, Qtune uses a double-state deep deterministic policy gradient (DS-DDPG) model, which introduces additional overhead compared with CDBTune.

5.2 Index selection

The target of index selection is to select columns to build indexes for optimal performance with certain workloads and storage limitations. There are several challenges in this task. First, the number of potential indexes is large, because indexes can be built on multiple columns. Second, some indexes may interact with each other [101]. Third, it is not always beneficial to build more indexes, because inserts or updates require expensive operations on indexes.

The index selection problem has been studied for decades, with typical research work shown in Table 4 [56–68]. The column 'Multi-column' shows whether multi-column indexes are supported, which is commonly used in modern databases. Traditional techniques use greedy algorithm or heuristic search for pure benefit, such as Drop [56], AutoAdmin [57], Anytime [58], and Dexter, or benefit per storage, such as DB2Advis [59], Relaxation [60], CoPhy [61], and Extend [62]. Kossmann et al. [102] evaluated these algorithms with experiments on TPC-H, TPC-DS, and JOB, as well as cost and parameter analysis. Some recent work proposes to use RL for this task, such as Basu et al. [63], Sadri et al. [64,65], Sharma et al. [66], and Lan et al. [67]. While Ding et al. [68] proposed a learning-based API for comparison of execution plan cost, to replace the optimizer API in traditional index selection techniques.

Table 4 Summary of research on index selection

Optimization target	Research work	Multi-column	Technique
Pure benefit	Drop [56]	No	Iteratively drop from current solution
	AutoAdmin [57]	Yes	Greedy with full enumeration
	Anytime [58]	Yes	Greedy with seed configurations
	Dexter	Limited 2	Greedy with hypothetical indexes
Per storage benefit	DB2Advis [59]	Yes	Greedy with hypothetical indexes & random substitution
	Relaxation [60]	Yes	Relax operations on the union set of candidates
	Cophy [61]	Yes	Linear programming
	Extend [62]	Yes	Iteratively extend current solution
Reward function of RL	Basu et al. [63]	Yes	Least square policy iteration (LSPI)
	Sadri et al. [64, 65]	No	Deep Q-Learning
	Sharma et al. [66]	No	Deep Q-Learning
	Lan et al. [67]	Yes	Deep Q-Learning
Plan cost comparison	Ding et al. [68]	–	Query-based machine learning

5.2.1 *Pure benefit*

Drop [56]. Drop generates the candidate column set with all potential single-columns. For each iteration, it drops one candidate and evaluates the workload processing cost, until a given index number is reached. Only single-column indexes are supported in this approach.

AutoAdmin [57]. AutoAdmin is proposed for Microsoft SQL Server. It generates the candidate column set with the union of candidates from all queries. It enumerates all possible subsets of a given size m, and uses a greedy algorithm to find $n > m$ indexes. It creates $(m+1)$-column indexes using m-column indexes by iterations.

Anytime [58]. Anytime is a refined version of AutoAdmin in the database engine tuning advisor (DTA) for Microsoft SQL Server. It considers both single and multiple-column indexes when generating the candidate set. The core selection algorithm is still greedy enumeration, and it uses seed configurations to avoid full enumeration, which also considers the index interactions.

Dexter[7]. Dexter is an open-source tool for PostgreSQL. It first gathers the templates and runtime information of queries from the plan cache, and estimates their initial cost with EXPLAIN command. Then it creates hypothetical indexes for all potential indexes and uses the optimizer to generate plans and estimate the cost. The hypothetical indexes that are used in the plans make up of the candidate set, and those with the most significant cost-savings are selected as recommendations. Only single- and double-column indexes are supported in this approach.

5.2.2 *Per storage benefit*

DB2Advis [59]. DB2Advis is proposed for IBM's DB2. It uses the optimizer and hypothetical indexes to generate the candidate index set. Then it sorts the candidates by benefit-per-space in decreasing order, and performs greedy selection until the storage budget is reached. Finally, it randomly substitutes the selected indexes with other candidates to find lower costs due to index interactions.

Relaxation [60]. Relaxation first selects the optimal indexes for each query, and unions them as the candidate set for the entire workload. Then it performs relaxed operations to reduce storage consumption, including merging, splitting, prefixing, promoting (to clustered index), and removing.

Cophy [61]. CoPhy formulates the index selection problem as linear programming of binary integer program (BIP). It minimizes the cost with the constraints that each query has a unique index option, and the storage does not exceed the budget. Off-the-shelf BIP solvers can be used for the problem. CoPhy also enables an interactive paradigm that allows early termination and incremental exploration.

Extend [62]. Extend starts with an empty set and extends the solution until the storage budget is reached. First, the single-column index with the best benefit-per-storage is selected. Then, for each iteration, a new single-column index is added, or a selected index is extended by appending an attribute.

7) https://ankane.org/introducing-dexter.

5.2.3 *Evaluation*

According to the experiments of Kossmann et al. [102], for different workloads, budgets, and runtime requirements, different approaches may work best. Extend and Anytime have the best overall performance for both runtime and solution quality. With large budgets, AutoAdmin and DB2Advis are advisable for short time, and Relaxation needs a longer time for better results. For small sizes, CoPhy guarantees optimal solutions. Dexter is simple to use and Drop is easy to implement.

In each case of the experiments, both the performance and the runtime, i.e., the time to output index suggestions, with different budgets are compared between the algorithms.

The first case is based on the TPC-H benchmark with 10 as the scaling factor. For memory budgets from 0 to 10 GB, Extend and Anytime have the best performance. Relaxation is only worse for 3 types of queries. Dexter and Drop perform worse than others. If budgets are large, the runtime of Anytime increases a lot, while others almost remain constant (<25 s). CoPhy has the longest runtime of around 150 s.

The second case is based on TPC-DS with 10 as the scaling factor. It differs from TPC-H that there is not a single dominating table, so impactful indexes do not have to be that large. For all budgets, Extend and Anytime still have the best performance. DB2Advis performs well for small (<2 GB) and large (>8 GB) budgets, but badly for the medium. Relaxation performs well for large (>6 GB) budgets. Dexter performs the worst. The runtime of DB2Advis, Dexter, and AutoAdmin is short (a few seconds), while Relaxation is the slowest (around 60 min).

The third case is based on JOB. Extend and Anytime perform well for small (<2 GB) and large (>3.5 GB) budgets, but badly for the medium because fine-grained solutions are missing. DB2Advis and CoPhy perform well for small budgets. Relaxation still performs well for large budgets. Drop and AutoAdmin cannot recommend indexes for small budgets, but Drop works well for the medium. Dexter still performs the worst and can only recommend indexes between 6 GB and 8 GB budgets. The runtime of DB2Advis and Dexter is still short, while Relaxation is still the slowest (40 to 50 min).

5.2.4 *Other approaches*

Reinforcement learning. To formulate the index selection problem as RL, it needs to specify the state, action, transition model, and reward function. In most RL-based index selection research, the state is defined as the currently built indexes, and the action is defined as choosing an index to build. The reward function and the algorithm to find the optimal policy vary between different pieces of work.

Basu et al. [63] defined a cost function instead of reward, which is the sum of the cost of index modification and query execution. They built a Markov decision process [103] and used least square policy iteration (LSPI) [104] to find the policy. Sadri et al. [64, 65] defined the reward function of the i-th step as $\frac{\mathrm{Cost}_0-\mathrm{Cost}_i}{\mathrm{Cost}_0}$, where Cost_0 and Cost_i indicate the query execution cost at the initial and the i-th step. Sharma et al. [66] defined the reward as $\max(\frac{\mathrm{Cost}_0-\mathrm{Cost}_i}{\mathrm{Cost}_i}, 0)$, and Lan et al. [67] defined it as $\frac{\mathrm{Cost}_{i-1}-\mathrm{Cost}_i}{\mathrm{Cost}_0}$. All three approaches use deep Q-Learning (DQN) [105, 106], a value-based method, to find the optimal policy. They use binary one-hot encoding for states.

The optimization target of RL-based approaches is slightly different from the previously mentioned approaches. The RL algorithm follows a trace of actions, and maximizes the overall reward on the trace, while previous approaches only minimize the query execution cost. It remains to be discussed whether the optimization target of RL is reasonable for index selection.

Comparison API. Most index selection algorithms rely on the optimizer to estimate the cost of query plans, which can be inaccurate. Ding et al. [68] proposed to use a machine learning model for the cost comparison of two execution plans, as an approach to optimizing the index recommendation. The plans are vectorized with a few feature channels, which measure the amount of work and encode the structure. The machine learning model takes the difference between two plans as input, and outputs which plan has a lower cost.

5.3 View materialization

View materialization refers to storing the results of selected queries or sub-queries, and using them in other queries. It can improve the database performance by reducing redundant computation. Due to a large number of queries and the cost of time and storage, it is challenging to identify which views to materialize.

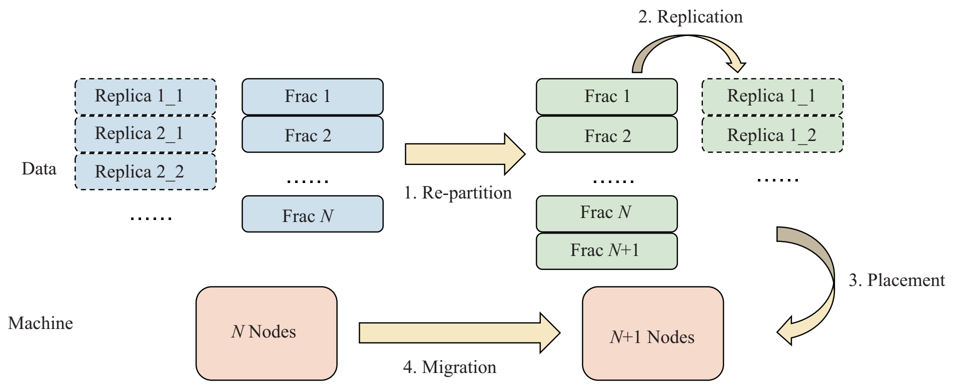

Figure 4 (Color online) Procedure of adding a node.

Dokeroglu et al. [69]. Dokeroglu et al. aimed at exploiting common tasks of multiple queries and generating an optimal global query plan. They detected common tasks of the given query set, and initialized a global query plan for all queries. A cost model is built for the execution time of tasks based on the statistics of the database. Then they proposed four algorithms to optimize the global query plan, including branch-and-bound algorithm (B&B) [107], genetic algorithm (GA), hill climbing algorithm (HCA) [108], and hybrid genetic-hill climbing algorithm (Hybrid-GHCA) [109, 110].

B&B searches all candidate solutions and selects the best one. It ensures the optimal solution but the time complexity is high. GA first randomly generates individuals, i.e., global query plans, then performs the crossover and mutation operations. The crossover operator takes two individuals, splits each into two segments, and exchanges the second segment. The mutation operator randomly changes one point of an individual. The new individuals always replace the worst ones. HCA is a local search algorithm that visits the neighbors of the current solution to find better ones. Hybrid-GHCA combines GA and HCA to improve the solution quality.

DB2 Design Advisor [70]. DB2 Design Advisor takes candidate views from several sources, including all SELECT-FROM-WHERE-GROUP BY combinations in the queries, logical views defined by the user, and suggested views from multiple-query optimization (MQO). The DB2 UDB technique for MQO rewrites the queries with suitable compensation to construct common sub-expressions. The DB2 Design Advisor selects a subset of the candidates within a storage budget with a greedy algorithm for the highest performance improvement.

BIGSUBS [71]. BIGSUBS is an approximate algorithm proposed for view selection by Jindal et al. They formulated the view selection problem into an integer linear programming (ILP) problem, which maximizes the utility of materialized views with cost constraint. Then they mapped it to a bipartite graph labeling problem [111] and proposed the BIGSUBS algorithm to solve it. The goal of BIGSUBS is to assign 1-labels to the sub-expressions for the highest utility. The labels are initialized randomly, and changed with a probability based on the utility and the storage cost for several iterations.

CLOUDVIEWS [72]. CLOUDVIEWS is a computation reuse framework for Microsoft's SCOPE job service [112, 113]. It supports online workloads with a feedback loop for runtime statistics, and focuses on recurring workloads. Two approaches for view selection are considered, including top-k heuristics and the BIGSUBS algorithm [71] mentioned previously.

Reinforcement learning. Reinforcement learning is also adopted for view selection. Yuan et al. [73] proposed to use DQN [105, 106] to solve the ILP formulation of view selection, with a wide-deep model to estimate the benefit of a materialized view. Liang et al. [74] also proposed a DQN-based approach with a different reward function.

5.4 Elastic resource

The elastic resource helps improve database performance by growing and shrinking the number of fragments, replicas, and nodes as needed [78]. As an example, the basic procedure of adding a node is shown in Figure 4. First, the data are re-partitioned and replicas are decided [114]. Then, it is decided how to place the data on the nodes. Finally, the data is migrated between all the nodes according to the placement decisions. The procedures of adding several nodes at once or reducing nodes are similar.

Research work on the elastic resource is categorized into two classes. The first one focuses on the allocation of resources and data, including adjusting the number of nodes, data re-partition, replication, and placement. The second one focuses on live data migration, which aims at performing reconfiguration without significant performance degradation of the system.

5.4.1 *Allocation*

The allocation of resources and data improves database performance in several ways. First, it matches the resource demand of dynamic workloads while minimizing the cost [79]. Second, it balances the loads of the horizontally partitioned data on different nodes [115] with certain partition, replication and placement strategies [75–78].

Accordion [75] proposes a dynamic data placement system for partition-based DBMSs that support ACID transactions. Firstly, Accordion estimates the capacity of servers heuristically considering the affinity between partitions, which indicates the frequency partitions being accessed together by the same transaction, and the access frequency of data partitions. Then Accordion formulates the data partition placement problem as a Mixed Integer Linear Programming (MILP) problem. The objective of this MILP problem is to find a partition placement plan that minimizes the data migration size while satisfying the defined constraints, which can be solved using robust optimization tools. The database adjusts its partition placement scheme and cluster size adaptively by activating Accordion periodically. However, this work has a limitation in that its coarse-grained partition strategy may lead to workload imbalance if multiple hot tuples concentrate in one data partition.

E-Store [76] aims at keeping load balance in different nodes of shared-nothing distributed DMBS. It adopts a two-tiered partitioning idea instead of traditional single-tier hash and range partitioning strategies. E-Store constantly monitors the CPU utilization of each database partition. If an imbalance is detected, the access times of each tuple are monitored to identify hot tuples and cold tuples. Compared with cold tuples, hot tuples are partitioned fine-grained and allocated first. To simplify the optimization of partition plans, E-Store focuses on scenarios where all non-replicated tables of an OLTP database form a tree-schema based on foreign key relationships, which means that it does not need to consider the transactions that span multiple nodes.

Clay [77] extends the E-Store method to both tree-based schemas and more complex schemas with arbitrary foreign key relationships. Instead of migrating all tuples of a pre-partitioned static block together at once, Clay dynamically creates blocks of tuples to migrate among servers, balancing the load as well as minimizing distributed transactions and migration overhead. The dynamic blocks, called clumps in Clay, are created on-the-fly based on a heat graph that records detailed information about co-accesses of the same transaction among tuples. The edge weights and vertex weights in the heat graph represent the frequency of tuples being accessed together and individually respectively. In order to balance the workload without creating new distributed transactions, a clump is constructed by expanding from the hottest vertex to the neighboring vertex that is most frequently co-accessed with it iteratively.

P-Store [79] proactively reconfigures the database while the system still has sufficient resources to migrate data and serve client transactions concurrently without performance interference. Because reactive methods like E-Store and Clay reconfigure the system when overloads have already occurred, which results in further performance degradation in a short time, proactive methods are needed to overcome this deficiency. P-Store does so by using a time-series prediction technique, Sparse Periodic Auto-Regression (SPAR) [116], to predict future load of the database, and proposing a dynamic programming algorithm to generate a cluster reconfiguration and data migration plan. P-Store makes several assumptions on its plan: (1) workload prediction is accurate; (2) access time of each data partition is uniformly distributed; (3) data distribution of each partition is uniform; (4) transaction distribution of the workload does not change quickly; (5) database size does not change quickly; (6) the workload has few distributed transactions. P-Store improves on existing work on database elasticity by scaling proactively rather than reactively, but it does not manage skew as E-Store and Clay do.

NashDB [78] proposes an adaptive data partition and configuration framework for OLAP workloads. OLAP databases often expect that queries with a higher priority should experience relatively higher performance than ones with a lower priority. In order to support query prioritization, NashDB relies on query priority and accessed times to estimate the value of tuples. After that, it provides automatic data fragmentation, replication, and placement strategies to design data reconfiguration plans that can balance the demand for the fragment (the fragment values) with the cost of storing these fragments.

It also optimizes the transition strategy to minimize data transfer using Kuhn-Munkres algorithm and presents the MAX OF MINS algorithm to route data access requests.

5.4.2 *Live data migration*

Live data migration aims at transferring data between machines while continuously serving the incoming transactions, instead of the stop-and-copy approach that causes latency. It takes the reconfiguration plan of data as input, and performs the migration procedure.

Albatross [80] is a live data migration technique that applies to a multi-tenant database system for OLTP workloads. This approach leverages the semantics of the database to migrate data efficiently while maintaining minimal impact on tenant performance. The incoming transactions are executed on the source node during migration. Simultaneously, the state of the database cache and active transactions is iteratively synchronized to the destination node. Although Albatross is a real-time data migration approach, there is still a short stop-and-copy migration when the source node hands over the transactions to the destination. This downtime may not be acceptable to the system with SLAs.

Zephyr [81] uses phases of on-demand pull and asynchronous push of data for live migration in a shared-nothing transactional database. It avoids service downtime by letting transactions run on the destination node during data migration. Zephyr divides the data migration process into three stages: Init Mode, Dual Mode, and Finished Mode. In the Dual Mode, previous active transactions on the source node continue running on the source node, while incoming transactions are all executed on the destination node. If a transaction on a destination accesses data that are not transmitted yet, it pulls the data from the source node. But to maintain synchronization, once the migration is initiated, metadata on the source node (for example, the index used to track the migration progress) can not be changed until the migration is finished, which can cause transaction abort in some cases. Another limitation of Zephyr is that transactions are blocked while the destination node is pulling a data chunk from a source node. If the data chunk is frequently accessed by multiple transactions, it will cause many transactions to block in a short time, resulting in a significant drop in throughput in a deterministic database system. The above problem will become more serious in the larger size of the data chunk, and result in further degradation in performance.

Squall [82] supports fine-grained data partition of databases in addition to the mechanism of on-demand pull and asynchronous push of data. It allows the initial data chunk to be split into sub-chunks by a query, and tracks the process of data migration for all sub-chunks at a particular partition. In this way, the destination node can only pull the required range of data for incoming transactions, rather than the whole data chunk at once. The fine-grained data partition method alleviates the transaction blocking issues in Zephyr. Besides, Squall routes some incoming transactions to the source node if all data associated with the transaction still locates at the source partition. In this way, the high latency caused by the frequent pull data of the destination node at the beginning of the migration can be avoided. Squall has been integrated with E-Store on H-Store [117]. In principle, it could be also used to execute the transition plans created by NashDB.

MgCrab [83] executes incoming transactions both on the source and destination machines during the migration, which differs from previous studies that execute incoming transactions on either the source or destination machine. MgCrab proposes a two-phase background push mechanism to overcome the limitation of incoming transactions being blocked on the source node during migration that existed in previous jobs. MgCrab is applicable to any deterministic database system. By leveraging determinism, it reduces the complexity and expensive overhead of ensuring consistency. How to introduce the MgCrab mechanism into non-deterministic systems is still a problem worth thinking about.

5.5 Storage management

Distributed storage management techniques improve the performance of DBMSs by enhancing the ability of the storage engine. On the one hand, they overcome the drawbacks of the general-purpose distributed frameworks like Spark [96]. Although these frameworks enable high-performance computing, there are some shortcomings related to big data management, such as the pressure on garbage collection and the expensive recomputation for partition and index construction. On the other hand, they support the additional methods for specific data types. For example, trajectory data requires additional querying and analyzing methods, such as k-nearest neighbor queries. The storage engine must support these methods with high performance.

UlTraMan [84] is a unified platform for big trajectory data management and analytics. It enhances the abilities of Spark by providing flexible mechanisms for random access, serialization, local indexing, and runtime persistence. The support for random access is a fundamental design in UlTraMan. In Spark, only the MEMORY_ONLY storage level potentially supports random access, where the data are fully deserialized. In UlTraMan, a new storage level ON_KV is introduced, which uses Chronicle Map[8] instances to index the data items, i.e., with integers as key and the data items as values. The Chronicle Map executes serialization and deserialization automatically, and supports both sequential and random access. Based on these, UlTraMan provides a derived API, RandomAccessRDD, to support random data access on the data of MEMORY_ONLY or ON_KV storage levels. It also provides flexible operational interfaces, including ID query, range query, KNN, and DBSCAN clustering.

Dragoon [85] is a hybrid system of big trajectory data management for offline and online. It supports both historical and streaming data processing with a hybrid storage design, where the mutable RDD model, mRDD, is proposed. The mRDD model aims at supporting the frequent updates of the streaming data. Each data partition is shared among multiple mRDDs, which is called RDD Share. Based on RDD Share, three update strategies are provided. First, the newest-only strategy directly replaces the old mRDD with the new one. It works when only the latest version of the trajectory data is needed. Second, the share-append strategy places the new data on newly created partitions, and appends the partitions to the target mRDD through RDD Share. It works when additional trajectory data are appended to the history. Finally, the share-update strategy creates a new RDD by sharing unchanged partitions and copying the remaining by inserting new data. It works when the trajectory data needs to be modified. Dragoon avoids data inconsistencies through the RDD Mirror mechanism, which supports read/write permissions and locks better in the distributed environment. Based on the hybrid storage design, Dragoon supports hybrid analysis for both online and offline, which includes loading, pre-processing, management, and analysis stages. Experiments show that for offline analytics, Dragoon has a similar performance as UlTraMan, but decreases up to double storage overhead for editing. For online analytics, Dragoon has better performance and scalability than general-purpose frameworks like Spark and Flink.

5.6 SQL antipattern detection

The antipatterns of SQL include bad schema and bad queries. The detection of SQL antipatterns can assist the DBAs to locate the bad SQLs and rewrite them. To the best of our knowledge, current techniques still rely on manually designed rules for the detection of such antipatterns.

Shao et al. [86] characterizes 34 types of antipatterns with empirical study. They summarize 24 database-access performance antipatterns from 27 research studies and 1 book, and derive 10 new antipatterns from the bugs of 7 direct-accessing web applications. They give the root causes of the antipatterns, and categorize them as inefficient or unnecessary computation, inefficient data accessing, unnecessary data retrieval, database design problems, application-design trade-offs, and so on. They also give a literal description of the fix strategies, but the automatic fix program is not provided.

Khumnin et al. [87] developed a tool that uses Transact-SQL language[9] to query and analyze the database schema. It can detect schema antipatterns and give instructions on refactoring. It works in a semi-automated manner to assist the DBA instead of a fully-automated manner, because it requires the semantic of data for the antipattern detection. Supported antipatterns include cloning tables or columns, creating multiple columns, leaving out the constraints, always depending on one's parent, and one size fitting all. They give definitions, examples, and detection heuristic on these antipatterns, which uses the Transact-SQL language.

SQLCheck [88] is another tool for antipattern detection. It focuses on the antipatterns of both intra- and inter-queries. It supports 26 antipatterns, which are categorized as logical design, physical design, query, and data. It uses heuristic rules to detect them, and ranks them with a score function.

Soar[10] is an open-source tool for SQL analysis and optimization. One of its functions is to rewrite the bad queries. It employs a bunch of rewrite rules in a certain sequence, and each rule corresponds to a possible antipattern. If the condition of a rule is met, it performs the rewrite operation to generate an optimized query and corresponding explanations.

8) https://github.com/OpenHFT/Chronicle-Map.

9) https://technet.microsoft.com/en-us/library/ms189826(v=sql.90).aspx.

10) https://github.com/XiaoMi/soar.

6 Challenges and future work

There are still several challenges related to database performance optimization. We summarize some typical challenges as follows, with some future directions of research.

6.1 Performance prediction

Inaccurate estimations from the optimizer. Most existing techniques rely on cardinality and cost estimations from the optimizer. Refs. [7, 14] revealed that such estimation could be inaccurate, which has negative effects on the prediction result. Recently, machine learning techniques are proposed for cardinality and cost estimation tasks to improve accuracy, including both supervised and unsupervised approaches. Nevertheless, cardinality and cost estimation is still a challenging task due to the dynamic workload and complex data distribution.

Gap to optimization. There is still a gap from prediction to optimization. For example, the prediction technique can tell how many machines the system needs to be added, but it cannot tell when and how to add them. Greedily adding them at once may not be a good choice, because the overhead of data movement could influence the performance of the system and violate the SLA. Therefore, additional techniques are needed between prediction and optimization.

6.2 Performance anomaly diagnosis

Limited anomaly data. Anomaly diagnosis techniques usually suffer from limited available anomaly data. Significant performance regressions that result in complaints from customers seldom occur in a stable database service. Small regressions may appear more frequently for OLTP workloads, but finding such slow queries requires recording and analyzing the execution time of all queries, which is expensive for OLTP workloads. It is not wise to focus on long execution time only, because queries executed in high frequency may have strong effects on user experience, even if the execution time is very short. It is also difficult to manually label such data with specific anomaly types. Most performance anomalies do not produce warning logs, so DBAs usually spend much effort analyzing the queries and monitoring data based on experience. The limited amount of labeled data can cause several problems, including cold start and data skewness between different anomaly types.

Overhead of data collection. There are basically three sources of monitoring data: system, database, and logs. System data are acquired from the system status, which is not expensive unless they are collected too frequently. Database monitoring data are acquired from certain views in a database. When tables are many and have large sizes, table-wise queries could take unaffordable time costs and resources. Logging overhead is affordable in the warning mode. However, warning logs are usually insufficient for performance diagnosis, so log-based techniques require logging in the debug mode, which results in high overhead. For accurate and comprehensive diagnosis, it is plausible to collect as much data as possible. Nonetheless, the overhead of data collection is a significant limitation.

OLTP vs. OLAP. The difference between OLTP and OLAP workloads is notable in anomaly diagnosis. There are different anomaly symptoms, such as the execution time of slow queries and monitoring data of resources. However, different symptoms may arise from the same root causes, such as bad index design and lock contentions. To achieve high accuracy, diagnosis techniques should consider such differences. For hybrid transaction/analytical processing (HTAP), the diagnosis is challenging due to this issue.

6.3 Performance optimization

When to optimize. Most optimization techniques are triggered by some signals instead of continuous working because the optimization overhead may affect the performance of the service, and bad actions may harm the system stability. Some of them need human instructions on when to optimize, which requires human efforts and relies on the experience of the DBA. Another choice is automatic triggers, such as system states, throughput drop, and high resource consumption. Both methods may start the optimization after the occurrence of performance regressions. A plausible way is to optimize ahead of time, which requires performance prediction techniques. However, it is still challenging to combine optimization with a prediction as mentioned before.

Dynamic workload. The workload in real-life databases dynamically changes, which affects the effectiveness of optimization plans. Some techniques use the backups of databases to run the optimization algorithm for trial and error. The workload they use is usually sampled from recent histories. When the optimization plan is generated, the workload on the service system may have become quite different from the sample workload, and the plan may not be suitable for the current environment.

Another choice is to optimize the database online [98]. Such techniques take the current or predicted workload as the input and directly perform optimization steps on the service system. They can keep pace with the dynamic workload, but it may risk the safety of the service system. Hence, the performance of the service system should not violate SLA during and after the optimization steps. If the technique performed trial-and-error steps or made wrong decisions, the safety requirement would be violated. Additional rules may help reduce such risks, but the design of such rules still requires human experience and effort.

6.4 Other topics

In recent years, the topic of self-driving DBMSs [10] has become popular, where the DBMS tunes itself according to the workload and database environment. It differs from previously discussed techniques as it works as a complete system that does not require any human effort to optimize the performance.

Currently, the optimization components of database systems work separately. The tuners of different configurations, e.g., knobs, indexes, and views, do not consider interactions with one another during the tuning process. The prediction and diagnosis tools do not automatically guide tuners as a complete system. It still requires the operation and maintenance personnel to keep monitoring the whole system and decide on approaches to solve performance problems.

Some other topics also need to be explored. Nowadays, there are many modern designs of database components, such as learning-based optimizers, nonvolatile memory, and large-scale distributed engines [118]. From the perspective of the operation and maintenance personnel, the performance prediction, diagnosis, and tuning of these systems should be different from traditional DBMSs and remain to be studied in the future.

7 Summary

Performance prediction, diagnosis, and tuning are important techniques for database performance optimization from the perspective of the operation and maintenance personnel. These topics have been widely studied for years in the field of database research. The prediction techniques take the workload and configuration information as the input and estimate the throughput or latency. They are strongly related to resource consumption and may have restrictions on DBMS scenarios. The diagnosis techniques are categorized by input data, i.e., monitoring metrics, logs, or time metrics, which correspond to their abilities. The monitoring metrics may support general-performance anomalies or root-cause key performance indicator detection. The logs typically support system behavior analysis. Time metrics may support specific issues, such as performance bottlenecks or resource contentions. Tuning techniques correspond to the influencing factors of database performance, including knobs, indexes, views, resources, and SQL design. We do not discuss the design of the DBMS in this review because we basically follow the perspective of the operation and maintenance personnel. This survey reviews notable research work on these topics, which provides some guidance on performance improvement in database services.

Acknowledgements This work was supported by National Natural Science Foundation of China (Grant No. 61832001) and ZTE-PKU Joint Program (Grant No. HV-CN-20210407026).

References

1 Ross R B, Amvrosiadis G, Carns P, et al. Mochi: composing data services for high-performance computing environments. J Comput Sci Technol, 2020, 35: 121–144

2 Lan H, Bao Z, Peng Y. A survey on advancing the DBMS query optimizer: cardinality estimation, cost model, and plan enumeration. Data Sci Eng, 2021, 6: 86–101

3 Dong Z Y, Tang C Z, Wang J C, et al. Optimistic transaction processing in deterministic database. J Comput Sci Technol, 2020, 35: 382–394

4 Zhou X, Chai C, Li G, et al. Database meets artificial intelligence: a survey. IEEE Trans Knowl Data Eng, 2022, 34: 1096–1116

5 Cooper B F, Silberstein A, Tam E, et al. Benchmarking cloud serving systems with YCSB. In: Proceedings of ACM Symposium on Cloud Computing, 2010. 143–154

6 Alomari M, Cahill M J, Fekete A D, et al. The cost of serializability on platforms that use snapshot isolation. In: Proceedings of IEEE International Conference on Data Engineering, 2008. 576–585

7 Leis V, Gubichev A, Mirchev A, et al. How good are query optimizers, really? In: Proceedings of the VLDB Endowment, 2015. 204–215

8 Ma M, Yin Z, Zhang S, et al. Diagnosing root causes of intermittent slow queries in large-scale cloud databases. In: Proceedings of the VLDB Endowment, 2020. 1176–1189

9 Mozafari B, Curino C, Jindal A, et al. Performance and resource modeling in highly-concurrent OLTP workloads. In: Proceedings of ACM SIGMOD International Conference on Management of Data, 2013. 301–312

10 Pavlo A, Angulo G, Arulraj J, et al. Self-driving database management systems. In: Proceedings of Conference on Innovative Data Systems Research, 2017

11 Ma L, Zhang W, Jiao J, et al. MB2: decomposed behavior modeling for self-driving database management systems. In: Proceedings of ACM SIGMOD International Conference on Management of Data, 2021. 1248–1261

12 Ganapathi A, Kuno H A, Dayal U, et al. Predicting multiple metrics for queries: better decisions enabled by machine learning. In: Proceedings of IEEE International Conference on Data Engineering, 2009. 592–603

13 Akdere M, Çetintemel U, Riondato M, et al. Learning-based query performance modeling and prediction. In: Proceedings of IEEE International Conference on Data Engineering, 2012. 390–401

14 Wu W, Chi Y, Zhu S, et al. Predicting query execution time: are optimizer cost models really unusable? In: Proceedings of IEEE International Conference on Data Engineering, 2013. 1081–1092

15 Marcus R C, Papaemmanouil O. Plan-structured deep neural network models for query performance prediction. In: Proceedings of the VLDB Endowment, 2019. 1733–1746

16 Zhou X, Sun J, Li G, et al. Query performance prediction for concurrent queries using graph embedding. In: Proceedings of the VLDB Endowment, 2020. 1416–1428

17 Bach F R, Jordan M I. Kernel independent component analysis. J Mach Learn Res, 2002, 3: 1–48

18 Shawe-Taylor J, Cristianini N. Kernel Methods for Pattern Analysis. Cambridge: Cambridge University Press, 2004

19 Sun J, Li G. An end-to-end learning-based cost estimator. In: Proceedings of the VLDB Endowment, 2019. 307–319

20 Mozafari B, Curino C, Madden S. DBSeer: resource and performance prediction for building a next generation database cloud. In: Proceedings of Conference on Innovative Data Systems Research, 2013

21 Yoon D Y, Mozafari B, Brown D P. DBSeer: pain-free database administration through workload intelligence. In: Proceedings of the VLDB Endowment, 2015. 2036–2039

22 Ester M, Kriegel H, Sander J, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of International Conference on Knowledge Discovery and Data Mining, 1996. 226–231

23 Thomasian A. On a more realistic lock contention model and its analysis. In: Proceedings of IEEE International Conference on Data Engineering, 1994. 2–9

24 Scarselli F, Gori M, Tsoi A C, et al. The graph neural network model. IEEE Trans Neural Netw, 2009, 20: 61–80

25 Yoon D Y, Niu N, Mozafari B. DBSherlock: a performance diagnostic tool for transactional databases. In: Proceedings of ACM SIGMOD International Conference on Management of Data, 2016. 1599–1614

26 Liu P, Zhang S, Sun Y, et al. FluxInfer: automatic diagnosis of performance anomaly for online database system. In: Proceedings of IEEE International Performance Computing and Communications Conference, 2020. 1–8

27 Samariya D, Ma J. A new dimensionality-unbiased score for efficient and effective outlying aspect mining. Data Sci Eng, 2022, 7: 120–135

28 Dundjerski D, Tomasevic M. Automatic database troubleshooting of azure SQL databases. IEEE Trans Cloud Comput, 2022, 10: 1604–1619

29 Nagaraj K, Killian C E, Neville J. Structured comparative analysis of systems logs to diagnose performance problems. In: Proceedings of USENIX Symposium on Networked Systems Design and Implementation, 2012. 353–366

30 Glasbergen B, Abebe M, Daudjee K, et al. Sentinel: universal analysis and insight for data systems. In: Proceedings of the VLDB Endowment, 2020. 2720–2733

31 Dias K, Ramacher M, Shaft U, et al. Automatic performance diagnosis and tuning in oracle. In: Proceedings of Conference on Innovative Data Systems Research, 2005. 84–94

32 Kalmegh P, Babu S, Roy S. Analyzing query performance and attributing blame for contentions in a cluster computing framework. 2017. ArXiv:1708.08435

33 Mogul J C, Wilkes J. Nines are not enough: meaningful metrics for clouds. In: Proceedings of ACM Workshop on Hot Topics in Operating Systems, 2019. 136–141

34 Cao W, Gao Y, Lin B, et al. TcpRT: instrument and diagnostic analysis system for service quality of cloud databases at massive scale in real-time. In: Proceedings of ACM SIGMOD International Conference on Management of Data, 2018. 615–627

35 Pettitt A N. A non-parametric approach to the change-point problem. Appl Stat, 1979, 28: 126–135

36 Agrawal R, Imielinski T, Swami A N. Mining association rules between sets of items in large databases. In: Proceedings of ACM SIGMOD International Conference on Management of Data, 1993. 207–216

37 Kim M, Sumbaly R, Shah S. Root cause detection in a service-oriented architecture. In: Proceedings of ACM SIGMETRICS Performance Evaluation Review, 2013. 93–104

38 Bentley J L. Multidimensional binary search trees used for associative searching. Commun ACM, 1975, 18: 509–517

39 Kim B, Rudin C, Shah J A. The Bayesian case model: a generative approach for case-based reasoning and prototype classification. In: Proceedings of Conference and Workshop on Neural Information Processing Systems, 2014. 1952–1960

40 Xing W, Ghorbani A A. Weighted PageRank algorithm. In: Proceedings of IEEE Conference on Communication Networks and Services Research, 2004. 305–314

41 Neapolitan R E, et al. Learning Bayesian Networks. Upper Saddle River: Pearson Prentice Hall, 2004

42 Bernstein P A, Cseri I, Dani N, et al. Adapting Microsoft SQL server for cloud computing. In: Proceedings of IEEE International Conference on Data Engineering, 2011. 1255–1263

43 Han J, Jia T, Wu Y, et al. Feedback-aware anomaly detection through logs aware anomaly detection through logs for large for large-scale software systems scale software systems. ZTE commun, 2021, 19: 88–94

44 Heckerman D, Chickering D M, Meek C, et al. Dependency networks for inference, collaborative filtering, and data visualization. J Mach Learn Res, 2000, 1: 49–75

45 Pele O, Werman M. Fast and robust earth mover's distances. In: Proceedings of IEEE International Conference on Computer Vision, 2009. 460–467

46 Kalmegh P, Babu S, Roy S. iQCAR: inter-query contention analyzer for data analytics frameworks. In: Proceedings of ACM SIGMOD International Conference on Management of Data, 2019. 918–935

47 Storm A J, Garcia-Arellano C, Lightstone S, et al. Adaptive self-tuning memory in DB2. In: Proceedings of the VLDB Endowment, 2006. 1081–1092

48 Zhu Y, Liu J, Guo M, et al. BestConfig: tapping the performance potential of systems via automatic configuration tuning. In: Proceedings of ACM Symposium on Cloud Computing, 2017. 338–350

49 Duan S, Thummala V, Babu S. Tuning database configuration parameters with iTunes. In: Proceedings of the VLDB Endowment, 2009. 1246–1257

50 Aken D V, Pavlo A, Gordon G J, et al. Automatic database management system tuning through large-scale machine learning. In: Proceedings of ACM SIGMOD International Conference on Management of Data, 2017. 1009–1024

51 Fekry A, Carata L, Pasquier T F J, et al. To tune or not to tune? In search of optimal configurations for data analytics. In: Proceedings of ACM KDD Conference on Knowledge Discovery & Data Mining, 2020. 2494–2504

52 Kunjir M, Babu S. Black or white? How to develop an autotuner for memory-based analytics. In: Proceedings of ACM SIGMOD International Conference on Management of Data, 2020. 1667–1683

53 Zhang X, Wu H, Chang Z, et al. ResTune: resource oriented tuning boosted by meta-learning for cloud databases. In: Proceedings of ACM SIGMOD International Conference on Management of Data, 2021. 2102–2114

54 Zhang J, Liu Y, Zhou K, et al. An end-to-end automatic cloud database tuning system using deep reinforcement learning. In: Proceedings of ACM SIGMOD International Conference on Management of Data, 2019. 415–432

55 Li G, Zhou X, Li S, et al. Qtune: a query-aware database tuning system with deep reinforcement learning. In: Proceedings of the VLDB Endowment, 2019. 2118–2130

56 Whang K. Index selection in relational databases. In: Proceedings of Foundations of Data Organization, 1985. 487–500

57 Chaudhuri S, Narasayya V R. An efficient cost-driven index selection tool for Microsoft SQL server. In: Proceedings of the VLDB Endowment, 1997. 146–155

58 Chaudhuri S, Narasayya V. Anytime algorithm of database tuning advisor for Microsoft SQL server. 2020. https://www.microsoft.com/en-us/research/publication/anytime-algorithm-of-database-tuning-advisor-for-microsoft-sql-server/

59 Valentin G, Zuliani M, Zilio D C, et al. DB2 advisor: an optimizer smart enough to recommend its own indexes. In: Proceedings of IEEE International Conference on Data Engineering, 2000. 101–110

60 Bruno N, Chaudhuri S. Automatic physical database tuning: a relaxation-based approach. In: Proceedings of ACM SIGMOD International Conference on Management of Data, 2005. 227–238

61 Dash D, Polyzotis N, Ailamaki A. CoPhy: a scalable, portable, and interactive index advisor for large workloads. In: Proceedings of the VLDB Endowment, 2011. 362–372

62 Schlosser R, Kossmann J, Boissier M. Efficient scalable multi-attribute index selection using recursive strategies. In: Proceedings of IEEE International Conference on Data Engineering, 2019. 1238–1249

63 Basu D, Lin Q, Chen W, et al. Regularized cost-model oblivious database tuning with reinforcement learning. In: Transactions on Large-Scale Data- and Knowledge-Centered Systems XXVIII, 2016. 28: 96–132

64 Sadri Z, Gruenwald L, Leal E. DRLindex: deep reinforcement learning index advisor for a cluster database. In: Proceedings of ACM Symposium on International Database Engineering & Applications, 2020. 1–8

65 Sadri Z, Gruenwald L, Leal E. Online index selection using deep reinforcement learning for a cluster database. In: Proceedings of IEEE International Conference on Data Engineering Workshops, 2020. 158–161

66 Sharma A, Schuhknecht F M, Dittrich J. The case for automatic database administration using deep reinforcement learning. 2018. ArXiv:1801.05643

67 Lan H, Bao Z, Peng Y. An index advisor using deep reinforcement learning. In: Proceedings of International Conference on Information and Knowledge Management, 2020. 2105–2108

68 Ding B, Das S, Marcus R, et al. AI meets AI: leveraging query executions to improve index recommendations. In: Proceedings of ACM SIGMOD International Conference on Management of Data, 2019. 1241–1258

69 Dökeroglu T, Bayir M A, Cosar A. Robust heuristic algorithms for exploiting the common tasks of relational cloud database queries. Appl Soft Computing, 2015, 30: 72–82

70 Zilio D C, Zuzarte C, Lightstone S, et al. Recommending materialized views and indexes with the IBM DB2 design advisor. In: Proceedings of International Conference on Autonomic Computing, 2004. 180–187

71 Jindal A, Karanasos K, Rao S, et al. Selecting subexpressions to materialize at datacenter scale. In: Proceedings of the VLDB Endowment, 2018. 800–812

72 Jindal A, Qiao S, Patel H, et al. Computation reuse in analytics job service at Microsoft. In: Proceedings of ACM SIGMOD

International Conference on Management of Data, 2018. 191–203

73 Yuan H, Li G, Feng L, et al. Automatic view generation with deep learning and reinforcement learning. In: Proceedings of IEEE International Conference on Data Engineering, 2020. 1501–1512

74 Liang X, Elmore A J, Krishnan S. Opportunistic view materialization with deep reinforcement learning. 2019. ArXiv:1903.01363

75 Serafini M, Mansour E, Aboulnaga A, et al. Accordion: elastic scalability for database systems supporting distributed transactions. In: Proceedings of the VLDB Endowment, 2014. 1035–1046

76 Taft R, Mansour E, Serafini M, et al. E-Store: fine-grained elastic partitioning for distributed transaction processing systems. In: Proceedings of the VLDB Endowment, 2014. 245–256

77 Serafini M, Taft R, Elmore A J, et al. Clay: fine-grained adaptive partitioning for general database schemas. In: Proceedings of the VLDB Endowment, 2016. 445–456

78 Marcus R, Papaemmanouil O, Semenova S, et al. NashDB: an end-to-end economic method for elastic database fragmentation, replication, and provisioning. In: Proceedings of ACM SIGMOD International Conference on Management of Data, 2018. 1253–1267

79 Taft R, El-Sayed N, Serafini M, et al. P-Store: an elastic database system with predictive provisioning. In: Proceedings of ACM SIGMOD International Conference on Management of Data, 2018. 205–219

80 Das S, Nishimura S, Agrawal D, et al. Albatross: lightweight elasticity in shared storage databases for the cloud using live data migration. In: Proceedings of the VLDB Endowment, 2011. 494–505

81 Elmore A J, Das S, Agrawal D, et al. Zephyr: live migration in shared nothing databases for elastic cloud platforms. In: Proceedings of ACM SIGMOD International Conference on Management of Data, 2011. 301–312

82 Elmore A J, Arora V, Taft R, et al. Squall: fine-grained live reconfiguration for partitioned main memory databases. In: Proceedings of ACM SIGMOD International Conference on Management of Data, 2015. 299–313

83 Lin Y, Pi S, Liao M, et al. MgCrab: transaction crabbing for live migration in deterministic database systems. In: Proceedings of the VLDB Endowment, 2019. 597–610

84 Ding X, Chen L, Gao Y, et al. UlTraMan: a unified platform for big trajectory data management and analytics. In: Proceedings of the VLDB Endowment, 2018. 787–799

85 Fang Z, Chen L, Gao Y, et al. Dragoon: a hybrid and efficient big trajectory management system for offline and online analytics. VLDB J, 2021, 30: 287–310

86 Shao S, Qiu Z, Yu X, et al. Database-access performance antipatterns in database-backed web applications. In: Proceedings of IEEE International Conference on Software Maintenance and Evolution (ICSME), 2020. 58–69

87 Khumnin P, Senivongse T. SQL antipatterns detection and database refactoring process. In: Proceedings of IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2017. 199–205

88 Dintyala P, Narechania A, Arulraj J. SQLCheck: automated detection and diagnosis of SQL anti-patterns. In: Proceedings of ACM SIGMOD International Conference on Management of Data, 2020. 2331–2345

89 Ge J K, Chai Y F, Chai Y P. WATuning: a workload-aware tuning system with attention-based deep reinforcement learning. J Comput Sci Technol, 2021, 36: 741–761

90 Sullivan D G, Seltzer M I, Pfeffer A. Using probabilistic reasoning to automate software tuning. In: Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems, 2004. 404–405

91 Zhang X, Chang Z, Li Y, et al. Facilitating database tuning with hyper-parameter optimization: a comprehensive experimental evaluation. In: Proceedings of the VLDB Endowment, 2022. 1808–1821

92 Tian W, Martin P, Powley W. Techniques for automatically sizing multiple buffer pools in DB2. In: Proceedings of Conference of the Centre for Advanced Studies on Collaborative Research, 2003. 294–302

93 Narayanan D, Thereska E, Ailamaki A. Continuous resource monitoring for self-predicting DBMS. In: Proceedings of IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2005. 239–248

94 Hutter F, Hoos H H, Leyton-Brown K. Sequential model-based optimization for general algorithm configuration. In: Proceedings of International Conference on Learning and Intelligent Optimization, 2011. 507–523

95 McKay M D. Latin hypercube sampling as a tool in uncertainty analysis of computer models. In: Proceedings of Conference on Winter Simulation, 1992. 557–564

96 Zaharia M, Chowdhury M, Das T, et al. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of USENIX Symposium on Networked Systems Design and Implementation, 2012. 15–28

97 Li Y, Shen Y, Zhang W, et al. OpenBox: a generalized black-box optimization service. In: Proceedings of ACM SIGKDD Conference on Knowledge Discovery & Data Mining, 2021. 3209–3219

98 Zhang X, Wu H, Li Y, et al. Towards dynamic and safe configuration tuning for cloud databases. In: Proceedings of International Conference on Management of Data, 2022

99 Lillicrap T P, Hunt J J, Pritzel A, et al. Continuous control with deep reinforcement learning. In: Proceedings of the International Conference on Learning Representations, 2016

100 Bellman R E. A Markov decision process. J Math Fluid Mech, 1957, 6: 679–684

101 Schnaitter K, Polyzotis N, Getoor L. Index interactions in physical design tuning: modeling, analysis, and applications. In: Proceedings of the VLDB Endowment, 2009. 1234–1245

102 Kossmann J, Halfpap S, Jankrift M, et al. Magic mirror in my hand, which is the best in the land? An experimental evaluation of index selection algorithms. In: Proceedings of the VLDB Endowment, 2020. 2382–2395

103 Puterman M L. Markov Decision Processes: Discrete Stochastic Dynamic Programming. New York: John Wiley & Sons, Inc., 1994

104 Lagoudakis M G, Parr R. Least-squares policy iteration. J Mach Learn Res, 2003, 4: 1107–1149

105 Mnih V, Kavukcuoglu K, Silver D, et al. Playing atari with deep reinforcement learning. 2013. ArXiv:1312.5602

106 Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning. Nature, 2015, 518: 529–533

107 Cosar A, Lim E, Srivastava J. Multiple query optimization with depth-first branch-and-bound and dynamic query ordering. In: Proceedings of International Conference on Information and Knowledge Management, 1993. 433–438

108 Mitchell M, Holland J H, Forrest S. When will a genetic algorithm outperform hill climbing. In: Proceedings of Conference and Workshop on Neural Information Processing Systems, 1993. 51–58

109 Lozano M, Herrera F, Krasnogor N, et al. Real-coded memetic algorithms with crossover hill-climbing. Evolary Computation, 2004, 12: 273–302

110 Tao F, Feng Y, Zhang L, et al. CLPS-GA: a case library and Pareto solution-based hybrid genetic algorithm for energy-aware cloud service scheduling. Appl Soft Computing, 2014, 19: 264–279

111 Martella C, Logothetis D, Loukas A, et al. Spinner: scalable graph partitioning in the cloud. In: Proceedings of IEEE International Conference on Data Engineering, 2017. 1083–1094

112 Chaiken R, Jenkins B, Larson P, et al. SCOPE: easy and efficient parallel processing of massive data sets. In: Proceedings of the VLDB Endowment, 2008. 1265–1276

113 Zhou J, Bruno N, Wu M C, et al. SCOPE: parallel databases meet MapReduce. VLDB J, 2012, 21: 611–636

114 Ji Y, Chai Y, Zhou X, et al. Smart intra-query fault tolerance for massive parallel processing databases. Data Sci Eng, 2020, 5: 65–79

115 Mehta M, DeWitt D J. Data placement in shared-nothing parallel database systems. VLDB J, 1997, 6: 53–72

116 Chen G, He W, Liu J, et al. Energy-aware server provisioning and load dispatching for connection-intensive internet services. In: Proceedings of USENIX Symposium on Networked Systems Design and Implementation, 2012. 337–350

117 Kallman R, Kimura H, Natkins J, et al. H-store: a high-performance, distributed main memory transaction processing system. In: Proceedings of the VLDB Endowment, 2008. 1496–1499

118 Hao D, Luo S M, Zhang H S. A distributed in-memory database solution for mass data applications. ZTE Commun, 2020, 8: 45–48