Learned Distributed Query Optimizer: Architecture and Challenges

GAO Jun[1], HAN Yinjun[2], LIN Yang[2], MIAO Hao[1], XU Mo[2]
(1. Peking University, Beijing 100871, China;
2. ZTE Corporation, Shenzhen 518057, China)

DOI: 10.12142/ZTECOM.202402007
https://kns.cnki.net/kcms/detail/34.1294.TN.20240521.1932.002.html, published online May 23, 2024
Manuscript received: 2023-07-28

Abstract: The query processing in distributed database management systems (DBMS) faces more challenges, such as more operators, and more factors in cost models and meta-data, than that in a single-node DMBS, in which query optimization is already an NP-hard problem. Learned query optimizers (mainly in the single-node DBMS) receive attention due to its capability to capture data distributions and flexible ways to avoid hard-craft rules in refinement and adaptation to new hardware. In this paper, we focus on extensions of learned query optimizers to distributed DBMSs. Specifically, we propose one possible but general architecture of the learned query optimizer in the distributed context and highlight differences from the learned optimizer in the single-node ones. In addition, we discuss the challenges and possible solutions.

Keywords: distributed query processing; query optimization; learned query optimizer

Citation (Format 1): GAO J, HAN Y J, LIN Y, et al. Learned distributed query optimizer: architecture and challenges [J]. ZTE Communications, 2024, 22(2): 49 - 54. DOI: 10.12142/ZTECOM.202402007
Citation (Format 2): J. Gao, Y. J. Han, Y. Lin, et al., "Learned distributed query optimizer: architecture and challenges," *ZTE Communications*, vol. 22, no. 2, pp. 49 - 54, Jun. 2024. doi: 10.12142/ZTECOM.202402007.

1 Introduction

Distributed database management systems (DBMSs) serve as a key infrastructure to manage data when the storage and processing of data exceed the capability limitation of a single-node computer's node. The data are usually partitioned into different computer nodes according to different strategies, such as hash, range, and round-robin, and then the query is processed over the partitioned data transparently. Different kinds of distributed DBMS have emerged recently, including the distributed version of traditional databases[1-2] and new products in the distributed context[3-4]. In this paper, we assume that the distributed DBMS runs on homogenous hardware and software, which is more like parallel databases. We ignore the differences between the two terms in this paper.

Query processing is essential to the distributed DBMS. As in a single-node DBMS, the query processing in distributed DBMS hides its implementation details. When the end users submit a query, a distributed DBMS makes an evaluation plan according to its meta-data and statistics maintained, retrieves data from the different partitions, and shuffles data when needed. End users only need to express their query needs without considering the data placement or the detailed evaluation plans.

This work was partially supported by NSFC under Grant Nos. 61832001 and 62272008, and ZTE Industry-University-Institute Fund Project .

We can see that distributed query processing faces more challenges than its single-node version. First, the data are organized as partitions across the different computing nodes. Different partition strategies will impact the following query performances, and thus some kinds of statistics should be maintained to guide the query optimization at the granularity of partitions. Second, more operators are introduced in the distributed DMBS, including the operators to move the data across nodes, distributed versions for traditional operators, etc. These operators offer much larger space for optimization. Third, the cost model in the single-node DBMS should be substantially extended to consider other factors in the distributed context, like the communication cost and the computation skew. In fact, the evaluation cost in distributed DBMSs is largely determined by the slowest computer node[4].

The search space in the query optimization of the distributed DBMS includes the join order search, the physical operator selection, the movement of data, and the placement of operators on the computer nodes. The first two aspects, which are also present in a single-node DBMS query optimization, are recognized as NP-hard problems[5]. The latter two are related to the multiple computer nodes in distributed DBMSs. The data stored in the partitions have movement strategies different from other computer nodes, each of which is for various communication and computation costs[6]. Additionally, the placement of operators on computation nodes should be considered, as some pushing-down operators can reduce the inter-

mediate results.

One simple but usually effective method in query optimization of the distributed DMBS is the heuristic rule. That is, the plan is generated as in the single-node DMBS in the first phase. Then in the second phase, the distributed DMBS chooses the distributed version for the physical operators in their evaluation plan, in which the data are first shuffled to other nodes, the computation is performed in nodes, and the final results are collected from nodes. The heuristic optimizer can lower the cost in the optimization phase and always exploit the power of different computer nodes, which results in a competitive performance.

However, the heuristic optimizer is too rough for distributed query processing. Ref. [7] points out that the plan should be constructed as a whole, while the partitions are totally ignored in the heuristic plan generation. In addition, all physical operators involve the data movement in the first phase, whose communication cost may be reduced if one data movement can be shared by multiple operators. Moreover, the cost model should be extended to consider the communication and computation skewness. Finally, it is hard for these heuristic rules to adapt to new hardware in distributed DBMSs.

Learned query plan generation[8 - 14] has been studied and shows its advantages in the single-node DBMS. Most learned optimizers[9 - 13] view plan generation as a sequence of action decisions. They introduce encoders to represent a query expression, capture data statistics, and then rely on a reinforcement learning (RL) framework to learn the action policy or the value of states, which can be trained with the evaluation latency or cost of the generated query evaluation plan.

We also notice that generative models like GPT have achieved remarkable success in various tasks, which also brings important inspiration to the learned query plan generation. However, we cannot directly apply the GPT models, at least the current version, to generate the query plan. As a language model, GPT cannot yield the tree-structured plan directly. In addition, GPT lacks statistics data and meta data in distributed DBMSs, which is highly needed in efficient plan generation.

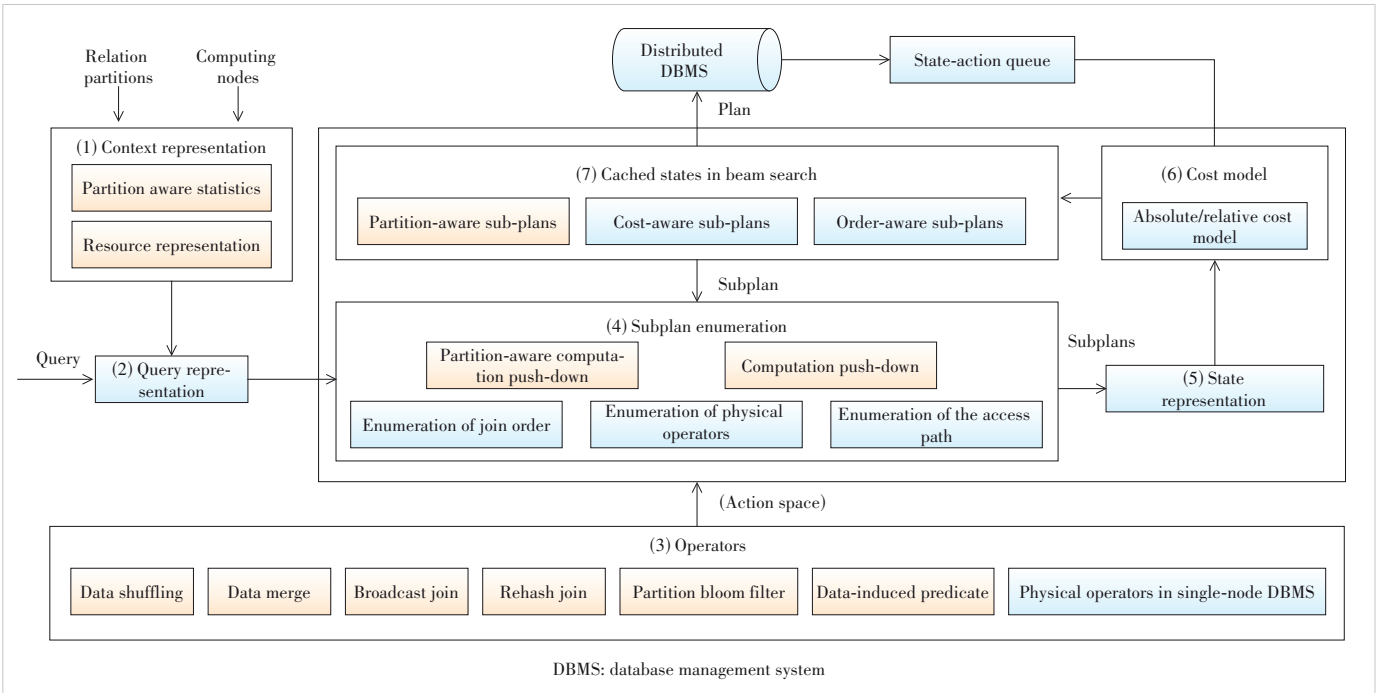
This paper mainly focuses on how to extend the learned optimizer in single-node DBMSs to distributed DBMSs. We first propose a possible but general architecture, then discuss challenges as well as possible solutions. We notice that there exist surveys[15 - 19] for the combination of AI and databases, ranging from learned index, learned data layout, database tuning, and plan generation, and there are some works[20 - 22] accelerating the query processing using new hardware in distributed DBMSs. Different from existing works, this paper focuses on the learned plan generation in the distributed context.

2 Architecture of Learned Distributed Query Optimizer

In this section, we first show the overall architecture of a learned query optimizer in the distributed DBMS, and then detail each component especially in aspects of distributed extensions compared with those in the single-node DBMS.

2.1 Overview

Our proposed architecture is similar to the ones presented in works like ROTS[9], LOGER[13], etc. As shown in Fig. 1, the



▲Figure 1. Architecture of a learned query optimizer in distributed DBMS

boxes with grey color indicate the related techniques that have been discussed in the single-node DBMS. It follows the RL framework to generate the final plan evaluation tree. The architecture fits the value-based RL models, and most of the components can be reused in the policy-based RL models. The states, actions, and rewards can be described as follows.

• The states are the intermediate forest of the sub-plan trees in plan generation. The initial state contains all table nodes, each for one sub-plan tree. The table embeddings come from (2) component. The table nodes also capture the data statistics modeled by (1) component. The state transition indicates that two sub-plan trees are merged. When the generation terminates, an entire evaluation tree is constructed. Due to the different order of tables in join operations and different placement of operators, the space of states is huge. Each state is encoded into a state representation in (5) and is further fed into the value model in (6) to measure which states have the potential to achieve high-performance results.

• The actions in RL indicate choosing two sub-plan trees and selecting one operator node to merge two sub-plan trees in each step to plan generation. The internal nodes in the evaluation tree are operators in relational algebra. Besides physical operators in the single-node DMBS, the distributed DBMS introduces operators to optimize the communication cost, skip data partitions, and operate among partitions in (3). As the action space is huge, the value-based RL can take the heuristic plan enumeration rules in (4) into consideration, which only produces the potential promising states for evaluation. The plan search can take a beam search strategy to keep more candidates in the plan enumeration in (7), even though the candidates are not with the minimal estimated cost at that time[12 - 13]. Such a strategy is in the same line with dynamic programming in SystemR, a well-known classic query optimizer.

• The rewards come from the feedback of distributed DBMS. As Structured Query Language (SQL) hints can specify the join order and physical operator, the final generated plan can be expressed using SQL hint first and then is submitted to the distributed DBMS for evaluation. The latency of the query plan can be collected as the rewards in the training of the value model. The rewards can take the form of the absolute query latency or the relative speedup compared with the latency of the plan using the default optimizer from the database.

2.2 Context Representation

The context representation component enables the query representation to be aware of the computing resource in different nodes and data distribution. The computing resource modeling and the statistics among the partitions are rarely studied in the single-node DBMS.

It is necessary to model computing resources as the slowest computing nodes that dominate the entire cost in distributed computing. The computing nodes in the distributed DBMS are interconnected by the network. Thus, the computing resource can be modeled as a graph, where each computing node has various computing and storage features, and the link is for the communication between nodes. The computing resource may be modeled with a graph neural network (GNN), in which the embedding of the computing nodes is generated.

For the modeling of intra-partition statistics, it can leverage the advanced techniques for the extensively studied cardinality estimation[23 - 28] in the single-node DBMS, which is roughly categorized into the data-driven[23 - 25] and query-driven[26 - 28] with different training signals. Note that, besides the machine learning based statistics, distributed DBMSs can build histograms or bloom filters in partitions, and these exact statistics enable distributed DBMSs to develop new operators, like data-induced predicates[29], to skip partitions in query evaluation. For the modeling of inter-partition statistics, the intra-partition data distribution representation can be attached to the computing nodes as one kind of feature and then the computing resource graph can learn the data distributions among the computing nodes.

2.3 Query Representation

The query representation component converts a query expression into a representation. Roughly, we have two methods to generate the query representation, namely based on the logical query graph and based on the plan from the database. Such a component is necessary to the distributed query optimizer, but shows a few differences with that in the single-node DBMS.

The logical query graph-based method is to build the query representation from a logical query graph, which is constructed from query expression. A logical query graph can contain table nodes with predicates as their features and edges with the join predicates, or takes the form of a heterogonous graph with table nodes, column nodes, predict nodes, etc., which is adopted by the Real Time Operating System (RTOS), LOGER. Then, a GNN is employed to learn the table embeddings, which will be fed into the following query generation components.

The plan-based method takes the plan generated by the default optimizer in the DBMS[30, 40]. QueryFormer[30] is a representative work that uses a transformer to model the plan. As the plan tree is deep, a virtual node is introduced to enable fast information exchanges among nodes in the plan. The plan from the database contains rich and easily-exploited features. However, the plan takes the tree form, and two tables that can be joined directly may be far in the tree, which may lead to some kinds of information loss compared with the logical query graph.

2.4 Physical Operators (Action Space)

The actions in RL indicate selecting one operator as an internal node and extending one or two sub-plan trees. The action space is related to the number of the physical nodes and

the join orders. Obviously, the more operators in the distributed DBMS, the larger the search space. Here, we mainly list some of the operators introduced for the distributed DBMS, including the communication-related operators, the join operators, and partition-specific operators.

The communication-related operator can be one class of operators. Recall that the distributed version for physical operators includes the data movement first and then computation next. We can extract the communication operators as the first-class operators, which enables more flexibility in the plan construction. For example, Ref. [31] devises a method to exchange the order of communication and computation. In addition, the data merger operator can move the data from one partition to another, which can then reduce the communication cost in the following join operations. In other words, opposite to the data partition, a data merge operator lowers the communication overhead at the cost of the computation skew.

The join operators can be divided into the intra-partition and inter-partition operators. The intra-partition join operators can take a similar way as those in the single-node DBMS, like by hash-based and nested loop, and merge join according to the size and statistics distribution of input data. The inter-partition join is also extensively studied in Ref. [32], including the hash join to handle the equal join predicates, fragment-and-replicate join[33] for general join predicates, and asymmetric fragment-and-replicate join when two join tables are skewed.

The partition-specific operators work on partitions. The partitions are units in the data storage, and the query performance can be improved if the partitions are skipped. The partition bloom filter is one operator which can quickly check whether the partition contains the data meeting the query conditions or not. Another operator is the data-induced predicate operator[29], which can derive new predicates on partitions from the query and partition-related statistics.

2.5 Plan Enumeration

The action space in the distributed DBMS is huge. We can then introduce the heuristic rules in the plan enumeration, which only produces the promising candidates and then improves the stability of the reinforcement learning. Note that the heuristic rules are easy to incorporate in the value-based RL, which is taken by most of the learned optimizers. The following heuristic strategies recently studied in the literature can be considered in the learned optimizer.

The placement of the data movement operation is considered in Ref. [31]. As mentioned above, the data movement (or shuffling) becomes an independent operator and can be commutative with other operators in the plan. For example, with the consideration of the existing partition scheme, the data movement can be placed earlier, which can then reduce the large intermediate results.

Computation push-down[34] is also considered in the plan generation. That is, instead of collecting data from partitions and then performing operations like aggregating/distinction over the collected data, we can perform operators first at the granularity of partitions, and then transfer the results to the next phase. The predicates evaluated early at the granularity of partitions can reduce the intermediate results, which usually reduces the communication cost in the following.

2.6 State Representation

The state representation component captures different features of a constructed sub-plan forest and converts it into an embedding state that will feed into the following value model to help choose the candidate actions. This component shares similar functionalities with that of a single-node DBMS, but also displays several differences.

The first difference is the form of the evaluation plan. The plan in the single-node DBMS always takes the form of a left-deep tree, which can allow index-loop join and support the pipeline evaluation. However, neither the index-loop join nor the pipeline is the key improvement in the distributed DMBS. In other words, there are more bushy structure plans in the distributed DBMS. Such a change may impact the design of the underlying representation model. For example, QueryFormer[30] introduces a virtual node to handle the deep tree, while this issue is not serious in the bushy form.

The second difference is the heterogeneous nature of operators. As we mentioned before, distributed DBMSs support more operators than single-node DBMSs, while different operators are with different features, like input/output (IO), CPU, and communication cost. Most of the plan representation models, like Tree-CNN[35], Tree-LSTM[36], and Transformer[37], handle the nodes with the shared features space, which should be extended to capture the heterogeneous nature of operators.

2.7 Value Model

The value model plays a crucial role in the value-based RL and has a similar functionality to the critic model in the policy-based RL, which can be used to determine which states can result in a better performance. For example, Oceanbase[38] points out that in some cases the computation push-down is not beneficial, which can be detected with the aid of the value model. The value model on the final plan is actually the cost model for the query evaluation plan. Although the cost model is more complex, the learned value model in the distributed DBMS is similar to the corresponding one in the single-node DBMS.

The cost model in distributed DBMS should consider more factors, including the IO cost, CPU cost, communication cost, and the data skew. Fortunately, the cost model[39] in the learned optimizer need not explicitly express the weights of different factors. The value model will be trained with the signals like the query latency. With more data trained, the weights of different factors can be learned in the cost model automatically.

One possible extension of the cost model is the choice of

the absolute or relative cost model. The absolute value model tries to mimic the latency for one query, and the relative cost model compares the values of two candidate plans. As a plan contains multiple nodes, and the relative cost model can capture the differences between two plans more easily, we guess that the relative cost model might be more suitable in the distributed DBMS.

2.8 Cached Sub-Plans in Plan Search

The cached plans are the candidates which need to be explored during the plan search. To cache one plan with the potential minimal cost is similar to the greedy search, which can be implemented efficiently at the cost of sub-optimal results. Thus, it needs to cache more candidates to gain performance improvement. The existing learned optimizers, like BALSA and LOGER, incorporate more cached sub-plans in beam search.

The progress in the distributed DBMS shows the different criteria in selecting sub-plans to cache. Besides the sub-plans with the minimal cost or preserving tuple orders, some distributed DBMSs, like Oceanbase[3], cache the sub-plans with interesting partitioning, which may avoid data partitioning in the following operators.

These plan caching strategies can be combined into the learned query optimizer easily using the beam search. The value model can help choose the sub-plan with the possible minimal cost. In addition, we can apply similar heuristic rules in distributed DBMSs to select the sub-plans with orders on specific attributes which could be useful for later operations. These plans are cached for future exploration using the beam search in learned query optimizers.

3 Challenges of Learned Distributed Query Optimizer

We identify two key challenges associated with the learned distributed query optimizer and discuss potential solutions to these issues.

3.1 Instability of Learned Query Optimizer

The learned query optimizer has a high chance to produce the plans with performance improvement when queries in the test set share the similarity with those in the training set. However, when the test query differs, the learned query optimizer may produce sub-optimal or bad plans. The stability of the learned query optimizer should be substantially enhanced before it can be deployed in real-life applications.

There are two possible solutions to the issue. BAO[10] mainly generates the candidate plans using existing DBMS optimizers with learned global parameters, and the bad performance is avoided with the help of the existing DBMSs. In fact, BAO has been extended to distributed DBMS[11]. However, such a method cannot produce plans from scratch and also is restricted by the capability of DBMSs.

An alternative solution is to extend the learned optimizer to produce both the candidate plan and its confidence in producing such a plan. When the confidence is lower than a given threshold, it directly relies on the DBMS to produce the final plan. In this way, plans with poor performance can be avoided.

3.2 High Training Cost of Learned Query Optimizer

The training of a learned query optimizer needs to search candidate plans from a huge plan space, in which each candidate is evaluated against the DBMS to obtain its latency. In addition, RL is notoriously hard to converge. In all, the training of a learning query optimizer is expensive.

Besides the transfer learning and meta-learning RL to improve the sample efficiency, an evolutionary algorithm (EA) could be one possible solution. In fact, EA methods have been adopted by PostgreSQL and can produce high-quality plans when the number of tables exceeds a given threshold. The existing research study, like learned concurrency control, also shows that EA algorithms could produce more effective results with less training cost than reinforcement learning[41].

4 Conclusions

The advance of distributed DBMS is required to incorporate the promising learned query optimizer. This paper outlines a possible architecture of the learned optimizer, mainly highlighting the differences from the learned optimizer in the single-node DBMS. In addition, this paper lists two major challenges and discusses their possible solutions.

References

[1] MUKHERJEE N, CHAVAN S, COLGAN M, et al. Distributed architecture of Oracle database in-memory [J]. Proceedings of the VLDB endowment, 2015, 8 (12): 1630 - 1641. DOI: 10.14778/2824032.2824061
[2] BLAKELEY J A, CUNNINGHAM C, ELLIS N, et al. Distributed/heterogeneous query processing in Microsoft SQL server [C]//The 21st International Conference on Data Engineering (ICDE'05). IEEE, 2005: 1001 - 1012. DOI: 10.1109/ICDE.2005.51
[3] YANG Z K, YANG C H, HAN F S, et al. OceanBase [J]. Proceedings of the VLDB endowment, 2022, 15(12): 3385 - 3397. DOI: 10.14778/3554821.3554830
[4] CHANG L, WANG Z W, MA T, et al. HAWQ: a massively parallel processing SQL engine in hadoop [C]//The 2014 ACM SIGMOD International Conference on Management of Data. ACM, 2014: 1223 - 1234. DOI: 10.1145/2588555.2595636
[5] IBARAKI T, KAMEDA T. On the optimal nesting order for computing *N*-relational joins [J]. ACM transactions on database systems, 9(3): 482 - 502. DOI: 10.1145/1270.1498
[6] RUPPRECHT L, CULHANE W, PIETZUCH P. SquirrelJoin: network-aware distributed join processing with lazy partitioning [J]. Proceedings of the VLDB endowment, 2017, 10(11): 1250 - 1261. DOI: 10.14778/3137628.3137636
[7] WANG G P. The optimization of query processing in Oceanbase 4.0. [EB/OL]. (2022-11-23) [2023-08-01]. https: //zhuanlan.zhihu.com/p/586113453
[8] MARCUS R, NEGI P, MAO H Z, et al. Neo: a learned query optimizer [J]. Proceedings of the VLDB endowment, 2019, 12(11): 1705 - 1718. DOI: 10.14778/3342263.3342644
[9] YU X, LI G L, CHAI C L, et al. Reinforcement learning with tree-LSTM for join order selection [C]//The 36th International Conference on Data Engineering

(ICDE). IEEE, 2020: 1297 – 1308. DOI: 10.1109/ICDE48307.2020.00116
[10] MARCUS R, NEGI P, MAO H Z, et al. BAO: making learned query optimization practical [C]//The 2021 International Conference on Management of Data. ACM, 2021: 1275 – 1288. DOI: 10.1145/3448016.3452838
[11] NEGI P, INTERLANDI M, MARCUS R, et al. Steering query optimizers: a practical take on big data workloads [C]//The 2021 International Conference on Management of Data. ACM, 2021: 2557 – 2569. DOI: 10.1145/3448016.3457568
[12] YANG Z H, CHIANG W L, LUAN S F, et al. Balsa: learning a query optimizer without expert demonstrations [C]//The 2022 International Conference on Management of Data. ACM, 2022: 931 – 944. DOI: 10.1145/3514221.3517885
[13] CHEN T Y, GAO J, CHEN H D, et al. LOGER: a learned optimizer towards generating efficient and robust query execution plans [J]. Proceedings of the VLDB endowment, 2023, 16(7): 1777 – 1789. DOI: 10.14778/3587136.3587150
[14] DOSHI L, ZHUANG V, JAIN G, et al. Kepler: robust learning for faster parametric query optimization [EB/OL]. [2023-08-01]. https://arxiv.org/pdf/2306.06798v2
[15] WANG W, ZHANG M H, CHEN G, et al. Database meets deep learning [J]. ACM SIGMOD record, 2016, 45(2): 17 – 22. DOI: 10.1145/3003665.3003669
[16] ZHOU X H, CHAI C L, LI G L, et al. Database meets artificial intelligence: a survey [J]. IEEE transactions on knowledge and data engineering, 2022, 34(3): 1096 – 1116. DOI: 10.1109/TKDE.2020.2994641
[17] LAN H, BAO Z F, PENG Y W. A survey on advancing the DBMS query optimizer: cardinality estimation, cost model, and plan enumeration [J]. Data science and engineering, 2021, 6(1): 86 – 101. DOI: 10.1007/s41019-020-00149-7
[18] CAI Q P, CUI C, XIONG Y Y, et al. A survey on deep reinforcement learning for data processing and analytics [J]. IEEE transactions on knowledge and data engineering, 2023, 35(5): 4446 – 4465. DOI: 10.1109/TKDE.2022.3155196
[19] ZHAO X Y, ZHOU X H, LI G L. Automatic database knob tuning: a survey [J]. IEEE transactions on knowledge and data engineering, 2023, 35(12): 12470 – 12490. DOI: 10.1109/TKDE.2023.3266893
[20] GUO C X, CHEN H, ZHANG F, et al. Distributed join algorithms on multi-CPU clusters with GPUDirect RDMA [C]//The 48th International Conference on Parallel Processing. ACM, 2019: 1 – 10. DOI: 10.1145/3337821.3337862
[21] GAO H, SAKHARNYKH N. Scaling joins to a thousand GPUs. [EB/OL]. [2023-08-01]. https://adms-conf.org/2021-camera-ready/gao_presentation.pdf
[22] PAUL J, LU S L, HE B S, et al. MG-join: a scalable join for massively parallel multi-GPU architectures [C]//International Conference on Management of Data. ACM, 2021: 1413 – 1425. DOI: 10.1145/3448016.3457254
[23] YANG Z H, LIANG E, KAMSETTY A, et al. Deep unsupervised cardinality estimation [EB/OL]. (2019-11-21) [2023-08-01]. http://arxiv.org/abs/1905.04278
[24] HILPRECHT B, SCHMIDT A, KULESSA M, et al. DeepDB: learn from data, not from queries! [EB/OL]. (2019-09-02) [2023-08-01]. http://arxiv.org/abs/1909.00607
[25] WANG J Y, CHAI C L, LIU J B, et al. FACE [J]. Proceedings of the VLDB endowment, 2021, 15(1): 72 – 84. DOI: 10.14778/3485450.3485458
[26] DUTT A, WANG C, NAZI A, et al. Selectivity estimation for range predicates using lightweight models [J]. Proceedings of the VLDB endowment, 2019, 12(9): 1044 – 1057. DOI: 10.14778/3329772.3329780
[27] LI B B, LU Y, KANDULA S. Warper: efficiently adapting learned cardinality estimators to data and workload drifts [C]//International Conference on Management of Data. ACM, 2022: 1920 – 1933. DOI: 10.1145/3514221.3526179
[28] NEGI P, WU Z N, KIPF A, et al. Robust query driven cardinality estimation under changing workloads [J]. Proceedings of the VLDB endowment, 2023, 16(6): 1520 – 1533. DOI: 10.14778/3583140.3583164
[29] KANDULA S, ORR L, CHAUDHURI S. Pushing data-induced predicates through joins in big-data clusters [J]. Proceedings of the VLDB endowment, 2019, 13(3): 252 – 265. DOI: 10.14778/3368289.3368292
[30] ZHAO Y, CONG G, SHI J C, et al. QueryFormer [J]. Proceedings of the VLDB endowment, 2022, 15(8): 1658 – 1670. DOI: 10.14778/3529337.3529349
[31] ZHANG H, YU J X, ZHANG Y K, et al. Parallel query processing: To separate communication from computation [C]//International Conference on Management of Data. ACM, 2022: 1447 – 1461. DOI: 10.1145/3514221.3526164
[32] POLYCHRONIOU O, SEN R, ROSS K A. Track join: distributed joins with minimal network traffic [C]//SIGMOD International Conference on Management of Data. ACM, 2014: 1483 – 1494
[33] STAMOS J W, YOUNG H C. A symmetric fragment and replicate algorithm for distributed joins [J]. IEEE transactions on parallel and distributed systems, 1993, 4(12): 1345 – 1354. DOI: 10.1109/71.250116
[34] YANG Y, YOUILL M, WOICIK M, et al. FlexPushdownDB: hybrid pushdown and caching in a cloud DBMS [J]. Proceedings of the VLDB Endowment, 2021, 14(11): 2101 – 2113
[35] ROY D, PANDA P, ROY K. Tree-CNN: a hierarchical deep convolutional neural network for incremental learning [EB/OL]. (2019-09-18) [2023-08-01]. http://arxiv.org/abs/1802.05800
[36] TAI K S, SOCHER R, MANNING C D. Improved semantic representations from tree-structured long short-term memory networks [EB/OL]. (2015-05-30) [2023-08-01]. http://arxiv.org/abs/1503.00075
[37] VASWANI A, SHAZEER N, PARMAR N, et al. Attention is all you need [C]//The 31st International Conference on Neural Information Processing Systems. ACM, 2017: 6000 – 6010. DOI: 10.5555/3295222.3295349
[38] YANG Z K, YANG C H, HAN F S, et al. OceanBase: a 707 million tpmC distributed relational database system [J]. Proceedings of the VLDB endowment, 2022, 15(12): 3385 – 3397. DOI: 10.14778/3554821.3554830
[39] SIDDIQUI T, JINDAL A, QIAO S, et al. Cost models for big data query processing: Learning, retrofitting, and our findings [EB/OL]. (2020-02-07) [2023-08-01]. http://arxiv.org/abs/2002.12393
[40] MARCUS R, PAPAEMMANOUIL O. Plan-structured deep neural network models for query performance prediction [EB/OL]. (2019-01-31) [2023-08-01]. http://arxiv.org/abs/1902.00132
[41] WANG J C, DING D, WANG H, et al. Polyjuice: high-performance transactions via learned concurrency control [EB/OL]. (2021-06-15) [2023-08-01]. http://arxiv.org/abs/2105.10329

Biographies

GAO Jun (gaojun@pku.edu.cn) received his BE and ME degrees in computer science from Shandong University, China in 1997 and 2000, and his PhD degree in computer science from Peking University, China in 2003. Currently he is a professor with the School of Computer Science, Peking University. His major research interests include web data management, graph data management and AI+DB.

HAN Yinjun is a senior engineer with ZTE Corporation. He has published multiple papers, obtained more than ten authorized patents, won multiple provincial and ministerial awards, and is a senior member of CCF. His main research interests include database systems and storage systems.

LIN Yang is a research and development engineer of ZTE Corporation. She received her master degree from Nanjing University of Science and Technology, China in 2017. Her research interests include query optimization, AI4DB and DB4AI.

MIAO Hao s a postgraduate student in the School of Computer Science, Peking University, China. His major research interests include graph neural network and AI+DB.

XU Mo is a research and development engineer of ZTE Corporation. He received his master degree from Monash University, Australia. His research interests include query optimization, AI4DB and database kernel development.