Improving Learning Based Cardinality Estimation Using Sampling Joins

Wenqiang Li

***Abstract*— Cardinality estimation is a crucial component of the query optimizer in database systems, which selects query plans based on the results of cardinality estimates and outputs them to the query executor. This study initially proposes a data-driven learning-based cardinality estimation method. This method constructs a probabilistic graph model based on database data, transforming cardinality estimation into a probability estimation of the variables within the model, and provides a lightweight modeling approach. Subsequently, through extensive comparative experiments, the proposed method in this paper is compared with other cutting-edge learning-based cardinality estimation methods across various dimensions, demonstrating the superior performance of the method in handling cardinality estimation problems.**

***Index Terms*—Database; Cardinality Estimation; Query Optimization; Multi-Table Joins.**

I. Introduction

Databases, as system software for storing, managing, and retrieving data, are one of the cornerstones of the information technology field[1]. They support various aspects of modern society, from everyday banking transactions and social media to enterprise-level customer relationship management and data analysis[2]. The performance of databases directly affects the speed of data retrieval and system response time, which in turn impacts user experience and the efficiency of business decision-making. Database performance optimization involves multiple aspects, including the way data is stored, the design of index structures, and the mechanisms for processing data queries and updates. Among these, query optimization is particularly critical, as efficient query processing can significantly reduce data retrieval time and alleviate server load, improving the overall processing capability and resource utilization of the system. The query optimizer in a database system is responsible for analyzing user-submitted queries, generating multiple potential execution plans, and selecting the most efficient one for execution[3]. The quality of this decision directly affects the speed of query execution and the efficient use of system resources, which in turn affects the performance and stability of the entire database system. In the case of large data volumes and complex queries, an excellent query optimizer can significantly reduce query response time and resource consumption, enhancing user experience and system throughput[4].

Cardinality estimation plays a central role in the query optimization process. Cardinality, or the estimated size of the result set, is a key input for the optimizer when formulating execution plans[5]. The optimizer requires this estimate to calculate the cost of different execution plans. For example, when deciding whether to use a hash join or a nested loop join for the join of two tables, the size of the cardinality is an important consideration[6]. If the cardinality estimate is too high or too low, it can lead the optimizer to choose a suboptimal query execution plan, resulting in decreased query execution efficiency and increased resource consumption. Therefore, accurate cardinality estimation is extremely important for the query optimizer, as it is directly linked to the optimization of database query performance and the efficient use of system resources.

However, achieving accurate cardinality estimation has been a longstanding challenge in the field of query optimization. Traditional methods such as histograms and sampling, while computationally simple and efficient, often struggle to accurately handle data skew and correlations between attributes in complex queries. In recent years, with the rise of machine learning, its powerful ability to model complex relationships between data has led researchers to turn their attention to the field where machine learning and cardinality estimation intersect, giving birth to learning-based cardinality estimation methods. These methods use machine learning models to learn the intrinsic relationships between data, providing more accurate cardinality estimates and assisting the query optimizer in making better decisions. Compared to traditional methods, the advantage of learning-based cardinality estimation methods lies in their robust modeling capabilities for data distribution and correlations between attributes. This not only makes the estimates more accurate but also enhances the ability to handle complex query scenarios, thereby effectively supporting the query optimizer in making more reasonable decisions.

II. Related Work

A. *Query Drive*

Query-driven cardinality estimation methods typically use records of historical query requests and the results of executing these requests to train models. These methods extract features from historical queries, including the structure of the queries, predicates, and execution plans. By fitting a mapping relationship between these features and the actual results, they capture the cardinality distribution under specific query patterns.

For example, the LWGT model[7] proposed by Dutt et al. uses both neural networks[8] and XGBoost[9] methods to achieve this fitting. The advantage of such methods is their

Manuscript received March 19, 2024

Wenqiang Li, School of Computer Science and Technology, Tiangong University, Beijing, China

ability to optimize for specific database workloads. For instance, if a database system primarily handles a certain type of query, the query-driven model can learn the patterns of these queries to provide more accurate estimates. However, this characteristic also requires a large amount of historical query data to train effective models, and the model's generalization ability becomes very limited for new or uncommon query patterns. Based on a similar idea, Kipf et al. implemented the MSCN model[10] based on a multi-set convolutional neural network, which represents queries as feature vectors containing three modules (table, join, and predicate modules). Each module is a two-layer neural network, and the outputs of different modules are connected and fed into the final output network, which is also a two-layer neural network. This complex model improves the model's estimation accuracy and generalization to some extent but also increases the difficulty of training.

Hayek proposed a method to improve estimation accuracy by learning the containment rate between queries. The "containment rate" refers to the proportion of one query result set within another query result set. To achieve this, they developed a deep learning model called CRN[11], which can represent query features in the form of sets and vectors. In the CRN model, the query pair (q1, q2) is first transformed into a vector representation, mainly involving the transformation of the original query's tables, joins, and column predicates into set forms. Then, through a neural network MLPi, the obtained vectors are transformed into a unique representative vector. Finally, the model uses these two vectors to predict the containment rate between queries.

In 2023, Kipf et al. from MIT, the team behind MSCN, proposed an improved Robust MSCN[12] to overcome the issue of MSCN's heavy reliance on workload stability. This method first simulates workload drift in queries involving unseen tables or columns by randomly masking parts of table or column features during the training process, forcing the model to rely on more stable features based on the latest database statistics for prediction. Secondly, it introduces a join bitmap, a technique that extends feature-based sampling through the idea of lateral information transfer to ensure feature consistency in join operations. This method improves the generalization ability of query-driven methods to some extent but still requires a large amount of query workload for training, and the difficulty of obtaining training data remains high.

Also from 2023, BICE[13], proposed by a team from the University of Electronic Science and Technology of China in collaboration with Aalborg University and the Huawei database team, designed a feature extractor with four sub-encoders to extract different types of information from the query plan tree. It uses graph embedding methods to encode joins and designs a parallel network for filters to improve encoding efficiency, and employs a compressor based on bidirectional LSTM to learn the encoded output and reduce the learning difficulty of the estimation model.

B. *Data Driven*

Data-driven cardinality estimation methods directly learn the probability distribution of these attributes from the data in the database. By transforming cardinality estimation into a problem of estimating the probability of attributes under specific predicates, they complete cardinality estimation. Their advantage is that they do not require input historical query workloads and can more comprehensively capture the potential relationships and distribution characteristics between data. Since they do not rely on historical query records, they can better generalize to new queries, especially in scenarios where data relationships are complex or data distribution changes frequently. However, these methods usually involve constructing complex machine learning models, such as deep neural networks, which means that data-driven methods generally have more complex models and higher training costs.

In 2019, Yang et al. proposed a novel cardinality estimation method using a deep autoregressive model to capture the multivariate distribution of relational tables to improve the accuracy of cardinality estimation[14]. The core advantage of this method is its unsupervised learning ability, which can accurately capture the relationships between column attributes in the database without the need for data labels, including range queries. To effectively handle multidimensional range queries, the study incorporated Monte Carlo integration techniques, which can effectively handle high-dimensional range queries.

In 2020, Hilprecht et al. proposed DeepDB[15], which includes the development of a new deep probabilistic model—Relational Sum-Product Networks (RSPNs). This model is capable of capturing the joint probability distribution of data in databases, reflecting correlations between attributes as well as the data distribution of individual attributes. The design of RSPNs allows it to provide answers for various user tasks at runtime, such as query answering and cardinality estimation. DeepDB also supports direct updates to the database, meaning that insert, update, and delete operations can be directly reflected in the model without the need for retraining. This update capability gives DeepDB a significant advantage in handling dynamically changing data.

NeuroCard[16] is also a learning-based cardinality estimation method based on a deep autoregressive model. It originates from Naru and extends it to multi-table join scenarios, developed by Yang Zongheng and his team at the University of California, Berkeley. It can capture the correlations between all tables in the database without relying on independence assumptions between tables or columns. The biggest feature of NeuroCard is that it covers the entire database with a single neural density estimator, using sampling from joins rather than computing complete joins to address the high training cost problem. With lossless column factorization technology, it decomposes columns into multiple sub-columns to reduce the size of the autoregressive model, allowing the model to remain practical even when facing columns with a large number of different values.

FACE[17] is a novel cardinality estimator proposed by Li Guoliang's team from Tsinghua University at the VLDB conference in 2022. It is based on the Normalizing Flow model and can effectively learn the joint distribution of data in relational databases without being limited by the query workload. A notable feature of FACE is its ability to simplify complex probability distributions into manageable forms, such as multivariate normal distributions, through continuous joint distribution transformations. This allows it to use probability density to accurately estimate the cardinality of SQL queries. Additionally, FACE effectively handles discrete and string data by designing de-quantization methods and string data encoding techniques.

III. Related Technologies

A. *Sum-Product Networks*

Sum-Product Networks (SPNs[18]n innovative deep learning architecture that combines the representational power of neural networks with the inferential efficiency of probabilistic graphical models. SPNs construct a directed acyclic graph where nodes can be either sum nodes or product nodes, representing complex probability distributions. This structure allows SPNs to handle high-dimensional data while maintaining model complexity and enabling efficient inference. In SPNs, sum nodes mix (i.e., weighted sum) the outputs from lower-level nodes, which can be seen as a mixture over variable sets, similar to mixture models in probabilistic models. Product nodes, on the other hand, represent the independence between variables through multiplication operations, which helps simplify the model's complexity and improve computational efficiency. The core advantage of SPNs lies in their ability to learn complex structures in the data without the need for intricate feature engineering.

B. *Sampling Joins*

Sampling Joins[19] a technique used in database queries to quickly estimate the cardinality of the result of a join operation. This method involves drawing a sample from the tables involved in the join and then performing the join operation on these samples to approximate the entire dataset's join result. Since only a subset of the data is processed, sampling joins can significantly reduce the use of computational resources and speed up query response times, making them particularly suitable for handling large datasets and complex queries.

In the context of cardinality estimation, sampling joins allow database systems to quickly obtain an approximation of a join result without sacrificing too much accuracy. The key to this method is how to draw a representative sample from the dataset and how to infer the join cardinality of the entire dataset based on the results of the sample join. To control the estimation error, specific sampling techniques and statistical methods may need to be employed.

The efficiency and accuracy of sampling joins depend on the sample selection strategy and the characteristics of the dataset. In practice, this method can be part of the database optimizer, helping the optimizer balance speed and accuracy in the selection of execution plans. Although sampling joins may not provide completely accurate cardinality estimates, they offer a practical compromise in many situations, enabling database systems to respond to user queries more quickly while maintaining relative accuracy in the estimated results.

IV. Lightweight data-driven learning cardinality estimation method

A. *cardninality estimation based on sum-product networks*

In the database context, we can consider the attributes of a data table as random variables, with each row of data representing the concrete values of these random variables. Based on this, we can use Sum-Product Networks (SPNs) to establish a database cardinality estimation model. When this model receives a query request, it can identify the attributes and predicates involved in the query, and the cardinality estimation for this query can be transformed into estimating the joint probability of these variables satisfying all predicates.

B. *Building sum-product networks from data tables*

Data Preparation: First, perform uniform sampling on tables with large data volumes to reduce the scale of the data table, thereby indirectly reducing the size of the final model and speeding up the modeling process.

Recursive Data Partitioning: Start by creating an empty Sum-Product Network structure with a root node set as either a sum or product node. From the root node, recursively partition the dataset and assign the resulting subsets to all child nodes of that node.For sum nodes, use clustering algorithms such as K-Means to divide the dataset into multiple clusters, with each cluster corresponding to a subtree, and calculate the weight of its corresponding subtree based on the proportion of the cluster to compute the mixture probability. For child nodes of sum nodes, if they do not meet the conditions for constructing leaf nodes, set them as product nodes.For product nodes, determine how to divide all attributes into several sets with low correlation by calculating the random dependency coefficients between attributes, and based on the attribute division, divide the data into multiple subsets and assign them to all child nodes of that node. For child nodes of product nodes, if they do not meet the conditions for constructing leaf nodes, set them as sum nodes.

Learning Variable Probability Distributions at Leaf Nodes: If a node in the above partitioning process corresponds to a data subset that meets the conditions for forming a leaf node, set it as a leaf node. In this study, the condition is: the data subset has fewer rows than a threshold T and has 1 attribute. The purpose of this setting is to fully divide the data into several low-correlation and suitable data subsets during the recursive partitioning process, making it easier to learn the probability distribution of the attribute on that subset.

For the tool to learn the probability distribution of single variables, this study chooses to implement it through histograms, as histograms can model both continuous and discrete variables, which is more suitable for scenarios with diverse data types like databases. Moreover, histograms have a faster modeling speed, aligning with the goal of this study to build a lightweight model.

C. *Sampling connectivity combined with a learning-based approach*

Bernoulli Sampling: Bernoulli sampling uniformly samples all tuples in the original table with the same probability. In Bernoulli sampling, the original table is sampled with a fixed probability p, and if two data tables are separately Bernoulli sampled and then joined, it may lead to a quadratic reduction in the number of rows in the joined table. That is, if each uniform sample occupies $1/p$ of the original table, then the number of rows in their joined table is only $1/p$^2 of the original joined table.

Universe Sampling: Universe sampling maps tuples in the table to the [0,1] interval using a hash function h and decides whether a row is included in the sample based on the sampling rate p. If the hash value is less than p, the row is included in the sample. This method can avoid the quadratic reduction in the number of output, and the sampled table using universe sampling often contains more correlations, which may lead to uneven sampling. Universe sampling is particularly useful in equi-joins because it ensures that if a row in one table is sampled, then the row with the same join key in the other table will also be sampled.

Stratified Sampling: The goal of stratified sampling is to ensure that the sample table does not miss minority groups, in other words, rows with fewer occurrences of a certain feature are not completely un-sampled. It defines groups (or strata) based on one or more columns (called stratification columns) and ensures that at least k rows are uniformly and randomly drawn from each group. When the number of rows in a group is less than k, all rows are retained. This method helps to maintain the diversity and representativeness of data in join queries.

Since different sampling methods have different suitable scenarios, relying solely on one sampling method may lead to unbalanced sampling results. Therefore, this paper chooses a hybrid sampling method SUBS[20]which combines the three sampling methods mentioned above into a hierarchical structure. The workflow is as follows: (1) For a data table T, first divide it into k groups: $\{G1, G2, \dots Gk\}$, select a column from the table (usually the join key), and a set of parameters $\{p1,\ p2,\ \dots\ pk,\ q1,\ q2,\ \dots\ qk\}$, and then define a hash function $h: U \rightarrow\ [0,1]$. (2) For each group, Gi, sample using its corresponding universe sampling rate pi. (3) Apply the Bernoulli sampling rate qi to the sampling results from step (2).

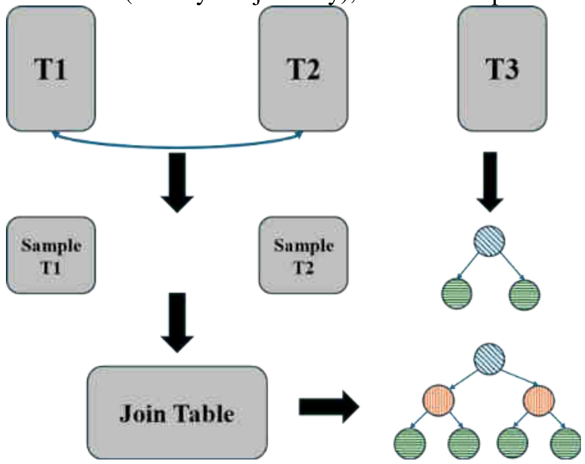

Fig.1 Sum-product network after combining sampled connections

When using Sum-Product Network models, we can pre-scan the metadata information of the database to obtain the dependency relationships between tables and perform the above hybrid sampling join on both sides of the dependency relationship, thus obtaining a smaller and more accurate joined table. Subsequently, build Sum-Product Network models on all single data tables and the sampled joined table. The overall workflow is shown in Figure 1, which illustrates a database with 3 data tables, where T1 and T2 have a foreign key dependency. Therefore, when creating the Sum-Product Network model, pre-sampling join was performed on T1 and T2 before entering the modeling stage. For the Sum-Product Network before joining the sampling, the strategy was to pre-create an anti-normalization join table, and if the size of the join table exceeded a certain threshold, it would be randomly sampled. After incorporating hybrid sampling join technology in this section, both the establishment time of the anti-normalization join table and the model construction time will be shortened, and naturally, the total volume of the model will also decrease.

V. EXPERIMENT

A. Test Benchmarks

This study adopts Synthetic as the test benchmark, which was initially derived from IMDB-JOB but was later modified by Kipf et al. The modifications included the removal of complex requests involving cyclic joins and string predicates, which are difficult for learning-based cardinality estimation to handle. A large number of query statements were randomly generated using templates, and this benchmark has since been widely used for benchmark testing in learning-based cardinality estimation. The benchmark utilizes the real-world IMDB dataset, which contains 5,000 query statements, with the majority of predicates being equality predicates and range predicates. The number of tables joined in join queries varies from 0 to 2.

B. subject of comparison

This research selected a mix of traditional and learning-based methods for comparison. The traditional method used is PostgreSQL version 13.1, while the learning-based methods include recent representatives of query-driven and data-driven approaches, such as Robust MSCN, BICE, DeepDB, and FLAT.

PostgreSQL: A representative of traditional cardinality estimation methods, PostgreSQL is a widely popular open-source database system that uses histograms and independence assumptions for cardinality estimation.

Robust MSCN: A query-driven learning-based cardinality estimation method, it is an improvement proposed by the MSCN authors to address the original method's inability to handle workload drift, which led to weak generalization capabilities. It is based on multi-set neural networks and incorporates techniques such as random query feature masking and join bitmaps during the training process.

BICE: A query-driven learning-based cardinality estimation method that employs four sub-encoders to extract features from the query plan tree and uses graph embedding and parallel networks for efficient encoding of joins and filters. It learns the encoder output of the feature extractor based on a bidirectional LSTM.

DeepDB and FLAT: Both are data-driven learning-based cardinality estimation methods with a core idea based on Sum-Product Networks. They propose RSPN and FSPN models, respectively, targeting multi-table join problems and strong attribute correlation problems.

C. Result analysis

1) Build efficiency

Table1 Build time comparison

modle	total	preparation	training
PostgreSQL	2min	0	2min
Robust MSCN	1687min	1630min	57min
BICE	1668min	1630min	38min
DeepDB	68min	17min	51min
FLAT	57min	12min	45min
SimpleBuild(ours)	40min	12min	28min

We posits that data importation should not be included in the data preparation phase of the model. Therefore, for

PostgreSQL, its modeling process can begin with the initiation of the ANALYZE command request and end upon the completion of the statement execution. During this process, PostgreSQL constructs multiple histograms at the system level to simulate the probability distribution of the data tables. Since it does not involve any complex calculations and only requires simple binning processing, the speed is very fast.

For Robust MSCN and BICE, both of which are data-driven learning-based methods, this study includes the training data preparation phase in the total duration of model construction. Both adopt the atomic MSCN training data generation scheme, obtaining query workload and actual cardinality as training data by executing 100K randomly generated query statements in PostgreSQL. Therefore, they require a lengthy data preparation process. However, thanks to the data-driven methods not needing to simulate the probability distribution of data, and both models using lightweight neural networks and LSTM, the training duration is relatively short.

As for DeepDB and FLAT, both are based on Sum-Product Networks, but DeepDB has a longer data preparation time because its RSPN model creates a large number of anti-normalization join tables during the preparation period to handle multi-table join requests. FLAT has a shorter training time compared to DeepDB, which is due to FLAT's leaf nodes not corresponding to just one attribute. To accelerate modeling, FLAT chooses to combine multiple related variables to jointly build multi-column histograms.

Regarding the SimpleBuild proposed in this study, since it simplifies the construction algorithm of the Sum-Product Network, it directly omits a large amount of clustering computation during the modeling process, significantly improving the modeling speed compared to DeepDB and FLAT, which also use Sum-Product Networks. In terms of total duration, the method proposed in this study is nearly 40 times faster in modeling speed compared to Robust MSCN and BICE, 1.7 times faster than DeepDB, and 1.43 times faster than FLAT.

Conclusion

This study begins by clarifying the problem definition of cardinality estimation and analyzes the strengths and weaknesses of traditional cardinality estimation methods as well as existing learning-based cardinality estimation methods. It concludes that the main work of this research is to propose a lightweight data-driven learning-based cardinality estimation method that can overcome the deficiencies in construction efficiency and access efficiency of data-driven learning-based cardinality estimation.

References

[1] Berg K L, Seymour T, Goel R. History of databases[J]. International Journal of Management & Information Systems (IJMIS), 2013, 17(1): 29-36.

[2] Taipalus T. Database management system performance comparisons: A systematic literature review[J]. Journal of Systems and Software, 2023: 111872.

[3] Leis V. Query processing and optimization in modern database systems[D]. Technische Universität München, 2016.

[4] Jarke M, Koch J. Query optimization in database systems[J]. ACM Computing surveys (CsUR), 1984, 16(2): 111-152.

[5] Harmouch H, Naumann F. Cardinality estimation: An experimental survey[J]. Proceedings of the VLDB Endowment, 2017, 11(4): 499-512.

[6] Han Y, Wu Z, Wu P, et al. Cardinality estimation in DBMS: a comprehensive benchmark evaluation[J]. Proceedings of the VLDB Endowment, 2021, 15(4): 752-765.

[7] 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.

[8] Rumelhart D E, Hinton G E, Williams R J. Learning representations by back-propagating errors[J]. nature, 1986, 323(6088): 533-536.

[9] Chen T, Guestrin C. Xgboost: A scalable tree boosting system[C]//Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. 2016: 785-794.

[10] Kipf A, Kipf T, Radke B, et al. Learned cardinalities: Estimating correlated joins with deep learning[J]. arXiv preprint arXiv:1809.00677, 2018.

[11] Hayek R, Shmueli O. Improved Cardinality Estimation by Learning Queries Containment Rates[C]//Proceedings of the 23rd International Conference on Extending Database Technology, EDBT 2020, Copenhagen, Denmark, March 30 - April 02, 2020. OpenProceedings.org, 2020: 157-168.

[12] Negi P, Wu Z, Kipf A, et al. Robust query driven cardinality estimation under changing workloads[J]. Proceedings of the VLDB Endowment, 2023, 16(6): 1520-1533.

[13] Liang Z, Chen X, Zhao Y, et al. Efficient Cardinality and Cost Estimation with Bidirectional Compressor-based Ensemble Learning[C]//2023 IEEE International Conference on Data Mining (ICDM). IEEE, 2023: 388-397.

[14] Yang Z, Liang E, Kamsetty A, et al. Deep Unsupervised Cardinality Estimation[J]. Proceedings of the VLDB Endowment, 13(3).

[15] Hilprecht B, Schmidt A, Kulessa M, et al. DeepDB: Learn from Data, not from Queries![J]. Proceedings of the VLDB Endowment, 13(7).

[16] Yang Z, Kamsetty A, Luan S, et al. NeuroCard: one cardinality estimator for all tables[J]. Proceedings of the VLDB Endowment, 2020, 14(1): 61-73.

[17] Wang J, Chai C, Liu J, et al. FACE: A normalizing flow based cardinality estimator[J]. Proceedings of the VLDB Endowment, 2021, 15(1): 72-84.

[18] Poon H, Domingos P. Sum-product networks: A new deep architecture[C]//2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops). IEEE, 2011: 689-690.

[19] Chen Y, Yi K. Two-level sampling for join size estimation[C]//Proceedings of the 2017 ACM International Conference on Management of Data. 2017: 759-774.

[20] Huang D, Yoon D Y, Pettie S, et al. Joins on Samples: A Theoretical Guide for Practitioners[J]. Proceedings of the VLDB Endowment, 13(4).