Cost-Based Lightweight Storage Automatic Decision for In-Database Machine Learning

Shuangshuang Cui[1], Hongzhi Wang[1,2(✉)], Haiyao Gu[1], and Yuntian Xie[1]

[1] Harbin Institute of Technology, Harbin, China
{20S103244,wangzh,1190201423,1181400603}@stu.hit.edu.cn
[2] Peng Cheng Laboratory, Shenzhen, China

Abstract. Storage structure decision for a database aims to automatically determine the effective storage structure according to the data distribution and workload. With the integration of machine learning and database becoming closer, complex machine learning tasks are directly executed in database, and need the support of efficient storage structure. The existing storage decision methods are mainly oriented to common workloads and rely on the decision of experienced DBAs, which has low efficiency and high risk of error. Thus, an automated storage structure decision method for in-database machine learning is urgently needed. We propose a cost-based lightweight row-column storage automatic decision system. To the best of our knowledge, this is the first storage structure selection for machine learning tasks. Extensive experiments show that the accuracy of the storage structure above 90%, shorten the task execution time by about 85%, and greatly reduce the risk of decision error.

Keywords: AI for DB · Row and column storage · Data partition

1 Introduction

In-database machine learning can lead to orders-of-magnitude performance improvements over current state-of-the-art analytics systems [1]. Complex machine learning tasks require efficient storage structures support. However, The existing decision of storage structure mainly depends on experienced DBAs [2], they make no quantitative analysis of the cost of the storage structures. It is difficult for them to make sure that their experience is correct, and there is a great risk of wasting resources and failing in the execution of tasks [3]. There is an urgent need for a solution that can provide automatic storage structure decision for DBAs and even ordinary database users. There are three challenges: (1) How to partition workloads to make feature extraction most efficient? (2) How to establish a cost model for the storage structure? (3) How to select features and collect relevant feature data efficiency?

– We propose a cost-based intelligent decision system for row and column storage for machine learning in database. To the best of our knowledge, this is the first storage structure selection system for machine learning (Sect. 2).

W. Zhang et al. (Eds.): WISE 2021, LNCS 13080, pp. 119–126, 2021.
https://doi.org/10.1007/978-3-030-90888-1_10

– We propose a data partitioning algorithm for machine learning task load in database, which is beneficial to increase the data scale in machine learning and improve the accuracy of the model. (Sect. 3.1)
– We propose a feature selection method for machine learning load and a storage engine performance acquisition algorithm, which can help inexperienced users to efficiently decide the appropriate storage structure. (Sect. 3.2 to 3.3)

2 System Overview

The core problem of automating the decision on the least cost storage structure is when and how to preprocess and extract features, and how to build the cost model and apply it. We use the read and write execution time of the workload as the cost parameter (Fig. 1).

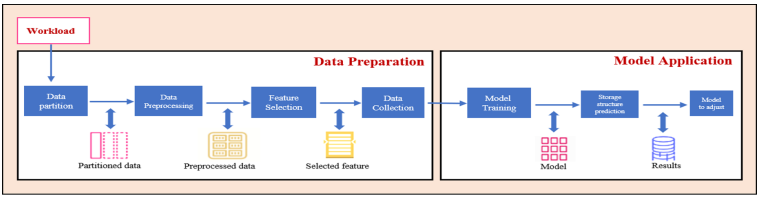


Fig. 1. Workflow of row-column storage intelligent decision method.

Architecture. The goal of the row and column storage decision system is to design the storage structure with the least cost for ML workload. The data preparation module mainly includes data partitioning and feature selection. The model application module is consists of module training and module adjusting.

Workflow. Too heavy machine learning workload can lead to efficiency problems. Thus, We first solve the data partitioning problem. As for the cost of the row and column storage structure, the key problem lies in the extraction of task features and generation of training data. We select five features that can most affect the execution efficiency and explain the reason. We also design a performance acquisition algorithm. Finally, the model is training.

3 Data Preparation

3.1 Data Partition

Since the excessive workload of machine learning in database, large-scale data will cause efficiency problems if preprocessed directly. The data partitioning algorithm can be specially oriented to non-uniformly distributed workload with addition of weighting factors for attributes, It can add weights to the workload according to the user's specific attributes.

The main function of Algorithm 1 is deviding large-scale data. The key idea of the algorithm is described as follows: First, get a certain m-dimension data

Algorithm 1. Data partition

input: T: the data table; K: the number of partition; $F_{i,j}$: the $j$the element of the file of i in F; a_n: the weighting factor;T_m:m dimension data selected from T;
Output: F:generated file;

```
1: n ← |T|
2: num ← n/k
3: T_m ← random(T, m)
4: for i : 1 → m do
5:     for j : 1 → n do
6:         T_m ← T_m * a_n
7:     end for
8: end for
9: for i : 1 → m do
10:    for j : 1 → num do
11:        F_{i,j} ← T_m[num * i + j]
12:    end for
13: end for
14: return F;
```

from the table to get the $n*m$-dimension matrix. Then, calculate the N data of each dimension of the m-dimension according to the weighting factor. According to the result of the weighted calculation, we can get the $n*1$-dimension weighted result T_m. Finally, sort the result and divide it evenly into k areas.

Algorithm Complexity. The algorithm line 4–8 is executed $m*n$ times, and the line 9–13 is executed $m*num$ times. So the total time complexity is $O(n)$.

3.2 Feature Selection

To select the features that represent the cost of the storage engine's selection, we analyzed and validated data patterns and workload-related characteristics. The influence of feature selection on performance prediction is shown in Table 1.

Table 1. Influence of feature selection on performance prediction.

feature / influence	keyFieldSize	nonKeyFieldSize	numOfFixedLengthField	numOfVarLengthField	numOfRows
The amount of data	√	√			
The amount of entries	√				
Database system processing			√	√	
Batch processing					√

(1) Key field size and non-key field size: The storage operates in blocks, and the data size of single row affects the amount of data in a block.
(2) the number of fixed-length fields and variable-length fields: the database system often has different processing methods for fixed-length fields and variable-length fields, which will affect the performance prediction.

(3) The number of rows involved in a single operation: The single-row amortized performance of a batch operation in the storage engine is higher than that of single-row operation.

3.3 Data Collection

The features we choose are the ones that most affect performance prediction, so it is more difficult to capture. We develop the storage engine performance data collection algorithm. Line 2–37 of the algorithm are executed in the row storage database. In line 3[1]–22, data schema is generated randomly and the related performance data required are calculated. For each selected field, its type is assigned randomly as fixed-length or variable-length. Its length is randomly generated according to the following formula: $x = x_0 + Exponential(\lambda)$, where x is the length to be defined for the field in the new table, and x_0 is the length of the field defined in the original table.

Line 11–15 and Line 20–24 calculate the key field size and non-key field size, the number of fixed length field, and the number of variable length field of the new data schema. Line 23 creates a new table based on the above information. Line 24–30 executes m insertions and records the average insertion time. Line 31–36 executes k random look-up query statements, recording the number of rows returned each time and query time.

Complexity Analysis. It is considered that the execution time of each query operation is $O(1)$, the total time complexity is $O(n^2)$.

4 Storage Decision Model

In the model application stage, there are two difficulties, i.e. how to establish a cost model for the collected performance data and select an appropriate model for training. We attempt to solve these problems in this section.

Cost Model. We use the performance data to train the cost model. We analyze the performance data collected above and design the cost model. For a given workload and data schema S, our model calculates the cost of row and column storage respectively as shown below:

$$Cost_{row}(S) = W_1 * V_{row-insert}(S) + W_2 * V_{row-select}(S)$$

$$Cost_{column}(S) = W_1 * V_{column-insert}(S) + W_2 * V_{column-select}(S)$$

Where W_1 denotes the number of insert in the workload, W_2 denotes the number of select in the workload, and V_x denotes the predicted value.

Model Training. The advantage of in-datdabase machine learning is efficient. It requires the training process of cost model to be efficient while ensuring accuracy. We chose XGBoost learners [4] to train the regression model, which is a tradeoff between performance and prediction accuracy for lightweight.

[1] a - total key field size; b - total non-key field size; c - the number of fixed-length fields; d - the number of variable-length fields.

Algorithm 2. Storage engine performance data acquisition

input: $Table$: original data; n: the number of schema to be generated; m: the number of insert; k: the number of select.

Output: $S_{row-insert}$: insert execution time in row; $S_{row-select}$: select e time in row storage; $S_{column-insert}$: insert time in column; $S_{column-select}$: select time in column.

$S_{row-insert}, S_{row-select}, S_{column-insert}$ *and* $S_{column-select} \leftarrow \emptyset$
for $i : 1 \rightarrow n$ **do**
  a, b, c *and* $d \leftarrow 0$;
  $D \leftarrow \emptyset$;
  $D \leftarrow D \cup Table.KeyFieldSet$;
  $p \leftarrow randomInt(0, |Table.NonKeyFieldSet|)$;
  $A \leftarrow random\ \ select\ \ p\ \ elements\ \ from\ \ Table.NonKeyFieldSet$;
  $D \leftarrow D \cup A$;
  for *each column* $C_i \in D$ **do**
    $C_{i.type} \leftarrow random\ \ select\ \ fromfixed - length, variable - length$
    if $C_{i.type} == fixed - length$ **then**
      $c \leftarrow c + 1$;
    else
      $d \leftarrow d + 1$;
    end if
    $C_{i.length} \leftarrow random\ \ select\ \ from\ \ [Table.getFieldLength(Ci), +\infty)$;
    if $C_i \in Table.KeyFieldSet$ **then**
      $a \leftarrow a + C_{i.length}$;
    else
      $b \leftarrow b + C_{i.length}$;
    end if
  end for
  *create new table Table*2 *with D*
  $timeInsert \leftarrow 0$;
  for $j : 1 \rightarrow m$ **do**
    insert Table.row$[j]$ *into Table*2;
    $timeInsert \leftarrow timeInsert + the\ \ execution\ \ time\ \ of\ \ line29$;
  end for
  $timeInsert \leftarrow timeInsert/m$;
  $S_{row-insert} \leftarrow S_{row-insert} \cup \{(a, b, c, d, 1, timeInsert)\}$
  for $j : 1 \rightarrow k$ **do**
    *random execute a select statement of SQL on Table*2;
    $timeSelect \leftarrow the\ \ execution\ \ time\ \ of\ \ line35$;
    $count \leftarrow the\ \ number\ \ of\ \ rows\ \ returned\ \ in\ \ line35$;
    $S_{row-select} \leftarrow S_{row-select} \cup \{(a, b, c, d, count, timeSelect)\}$
  end for
end for
*repeat line*2 *to line*37 *in column* $-$ *database*
return $S_{row-insert}, S_{row-select}, S_{column-insert}, S_{column-select}$;

Loss Function. The loss function adopts the common loss function of XGBoost model, and its general form is as follows:

$$Obj^{(t)} = \sum_{i=1}^{n} l(y_i, y_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + constant$$

The first term describes the training error, and the l function is used to measure the error between the predicted value and the true value. y_i is the true value, $y_i^{(t-1)}$ is the predicted value of the model obtained from the previous $t-1$ rounds of training, and $f_t(x_i)$ is the function to be trained in the t round.

$$\Omega(f) = \gamma T + 1/2\lambda\|\omega\|^2$$

Where T represents the number of leaf nodes, and ω represents the fraction of leaf nodes. The model training objective requires that the prediction error should be as small as possible.

Lightweight. To make the model more widely used, it is required that the model must be lightweight and migratable. We package the two proposed methods, and design the odbc interface, which can connect to database directly.

5 Experiments

5.1 Accuracy Evaluation

Table 2. Comparison of model accuracy.

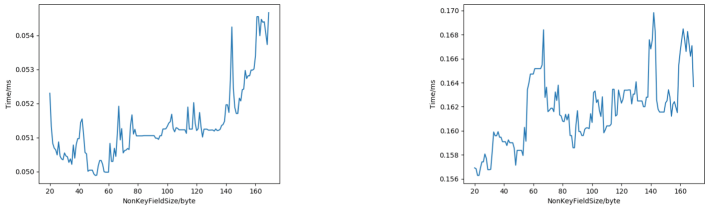
	Our model		Reference model	
	Insert	Select	Insert	Select
Row-oriented	**95.04%**	**91.40%**	**97.25%**	**90.07%**
Column-oriented	**92.18%**	**96.77%**	**92.37%**	**93.65%**

The experimental database is selected as OpenGauss1.1.0 [7]. The TPC-H public dataset was selected as benchmark. $accuracy = 1 - (V_{predict} - V_{truth})/V_{truth}$ [8], where the $V_{predict}$ is the time predicted. And the V_{truth} is the value of execution time of database feedback. The accuracy of our models are above 90% (Table 2). We compare the accuracy with the model proposed Wei et al. [5]. Under the row-oriented, the accuracy of our insert model is 2.21% lower than theirs. Because they use LSM storage engine, which has superior write performance.

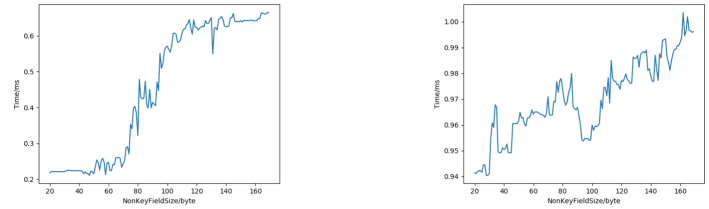
5.2 Feature Section Effectiveness Evaluation

We test five selected features to verify the validity of selected features. Due to the length of the paper, the process of verifying the influence of "non-key field length" on SQL execution time is listed. Under the premise of controlling a single variable, we set the key field length = 16, the number of fixed-length fields and variable-length fields = 5. We record the predicted value in row/column storage with the change of non-key field length. The results are shown in Fig. 2.

Although the predicted time of each models somewhat fluctuates, it generally increases with the augment of the length of non-key fields, because the storage engine operates on a block, and non-key fields affect the size of single row data. It affects the amount of data in a block which affects the execution time of the SQL in turn. It meets the expectation of feature selection, and the feature extraction time is the fastest while ensuring the model accuracy.



(a) insert under row model (b) select under row model



(c) insert under column model (d) select under column model

Fig. 2. The relationship between time predicted and non-key fields' length

5.3 Compare the Applicable Workloads of Row/Column Model

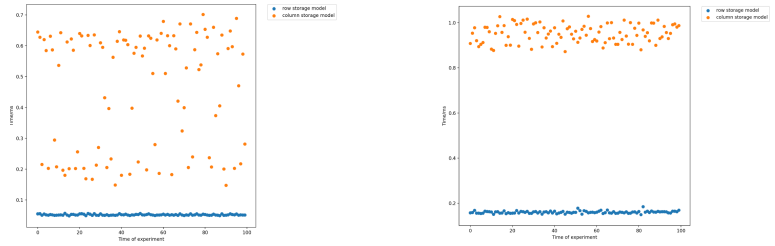


Fig. 3. Insert and select execution time predicted of row and column storage structure

We design the experiment repeats 100 times. Each time randomly generating data schema features, and comparing the predicted time values given by the row and column storage model.

The results (Fig. 3) show row storage require less execution time and predicted results more consistently under the vast majority of insertion and selection loads. This fully demonstrates the importance and necessity of analysing the storage cost quantitatively.

5.4 Comparisons on Various Workloads

The experimental results were normalized, and the experimental results were shown in Fig. 4.

Under transactional and transactional mixed workloads, row storage is selected according to experience, and the model recommendation structure is consistent with experience. Under analytical and analytical mixed workloads, the application of the model recommendation structure will result in approximately 85% performance improvement.

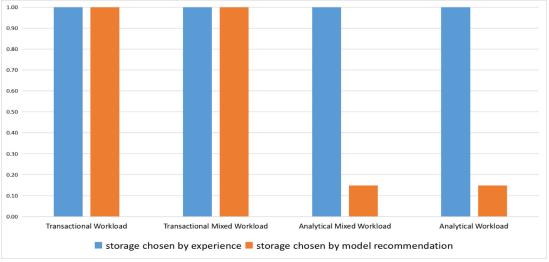


Fig. 4. Performance comparison between the storage structure selected in experience and the storage structure recommended by the model under different workloads.

6 Conclusions

This paper proposes a cost-based intelligent decision for row-column storage, so that the database can efficiently choose a storage structure suitable for the data even when the performance of the database is unknown. From experimental results, using the method proposed in this paper to determine the storage structure can shorten the task execution time by about 85%, greatly reduce the risk of decision errors, and greatly improve the efficiency of task execution. In the future, we plan to verify our proposed algorithm on more row and column storage databases to further enhance the generalization ability of the model.

Acknowledgements. This paper was supported by NSFC grant (U1866602, 71773025). The National Key Research and Development Program of China (2020YFB1006104).

References

1. Olteanu, D.: The relational data borg is learning. PVLDB **13**(12), 3502–3515 (2020)
2. De Marchi, F., Lopes, S., Petit, J.-M., Toumani, F.: Analysis of existing databases at the logical level: the DBA companion project. ACM SIGMOD Rec. **32**(1), 47–52 (2003)
3. Park, Y., Zhong, S., Mozafari, B.: Quicksel: quick selectivity learning with mixture models. In: Proceedings of the 2020 SIGMOD (2020)
4. Chen, T., Guestrin, C.: XGBoost: a scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD (2016)
5. Wang, H., Wei, Y., Yan, H.: Automatic storage structure selection for hybrid workload (2020)