WorthyPar: A Workload-Aware Data Hybrid Partitioning Advisor with Deep Reinforcement Learning

Shuangshuang Cui, Hongzhi Wang (✉), Jinghan Lin,
Xiaoou Ding, and Donghua Yang

Faculty of Computing, Harbin Institute of Technology, Harbin, China
{cuishuangs, linjinghan}@stu.hit.edu.cn,
{wangzh, dingxiaoou, yang.dh}@hit.edu.cn

Abstract. Data partitioning physically divides tables or databases to minimize I/O and maximize query processing performance. Designing subtle partitioning strategies for OLAP workloads is an important and challenging task. However, existing workload-aware partitioning strategies lack flexibility and fine-grained partitioning strategies lack adaptability. To address these limitations, we propose WorthyPar. To our knowledge, this is the first attempt to achieve self-driving hybrid partitioning relying on DRL. Specifically, we first demonstrate that the hybrid partitioning problem is NP-hard and formulate it as a Markov Decision Process (MDP) to train an automatic partitioning advisor. Subsequently, we propose a workload prediction model to forecast future workloads, and a performance analysis model to assess the performance of hybrid partitioning strategies without actual partitioning in the database. Extensive experiments demonstrate that WorthyPar can achieve up to 75% reduction in time.

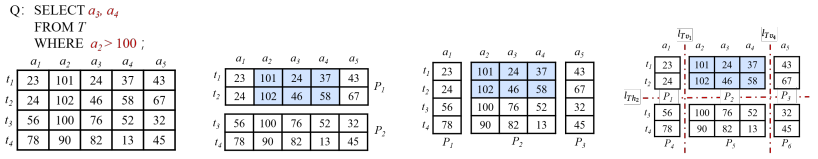
Keywords: Hybrid data partition · Workload forecasting · Deep reinforcement learning.

1 Introduction

In the era of big data, many scenarios require to process OLAP workloads efficiently [4, 7, 9, 17, 22]. OLAP workloads are dynamic, and their demands can change over time. This requires systems to not only handle queries efficiently but also adapt to dynamic workloads. For efficiency issues, data partition is crucial, which physically split a table or a database in order to maximize system performance [5, 6, 11]. However, finding subtle partitioning strategy for dynamic OLAP queries is a highly challenging problem. Even though many data partitioning strategies have been proposed due to its importance, they still have limitations, which can be summarized in two aspects:

1. Workload-aware partitioning strategies lack flexibility. Some data partitioning strategies using DRL [8,10] only horizontally or vertically partition the table based on the specified partition key. They are inflexible which cause

redundant data will be scanned when the query only requires selecting some of the columns with filter condition. A simple example for table T based DRL as shown in Fig. 1.



(a) Logical table T and example query Q

(b) Horizontally Partition the T using a_2 as the partitioning key to benefit Q

(c) Vertically Partition the T by combining a_2, a_3 and a_4 into a column group to benefit Q

(d) Hybrid Partition the T to benefit Q

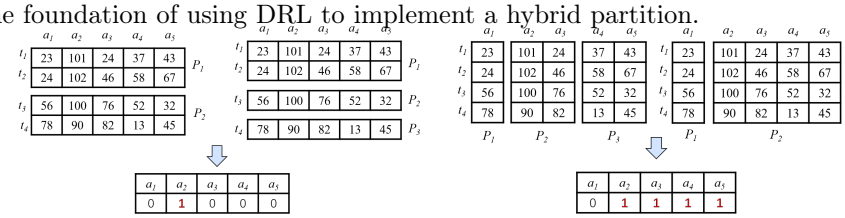
Fig. 1. An example of horizontally and vertical partition for table T based DRL.

Consider the table T and the query Q in Fig. 1(a). They horizontally partition T into P_1 and P_2 using a_2 as the partition key (shown in Fig. 1(b)). The blue shading represents the values accessed Q. Q will scan the relevant values of a_1 and a_5 as redundant data in P_1 when accessing a_2, a_3 and a_4. The core weakness of the horizontal partition is that it cannot place attributes of a tuple on different partitions, e.g., partition a_1 in P_a, partition a_2, a_3, and a_4 into P_b, and partition a_5 into P_c. Similarly, the vertical partition for table T also scan redundant data in P_2 when accessing a_2, a_3 and a_4 (shown in Fig. 1(c)). Obviously, if we use hybrid partition method, we can reduce the redundant data accessed (shown in Fig. 1(d)). Therefore, when workload-aware partitioning strategies aim to further reduce access to redundant data, adopting fine-grained hybrid partitioning strategies is a promising approach.

2. Fine-grained partitioning strategies lack adaptability. Existing fine-grained hybrid partitioning methods can be divided into two types. (1) Tables are first horizontally partitioned into sub-tables, and then optimal vertical partitioning strategies are applied within each sub-table. Since tuples in different column groups are aligned, there is no need for tuple alignment reconstruction, as in Peloton [13]. (2) Tables are first vertically partitioned into column groups, and then horizontal partitioning is applied within each column group. This disrupts tuple order and requires maintaining additional metadata for tuples in each block, leading to significant reconstruction overhead, as seen in GSOP [15]. These methods use machine learning to guide vertical partitioning decisions but have limited impact on query performance. Jigsaw [14] introduces an irregular partitioning method that eliminates redundant data but only works for static workloads. Current fine-grained hybrid partitioning methods lack adaptability because they neither predict workloads nor consider deployment costs. Fixed partitioning structures result in resource waste and performance bottlenecks. Therefore, for dynamic OLAP workloads, fine-grained hybrid partitioning must include adaptive capabilities.

As can be seen from the problem described above, existing advanced DRL-based automatic partitioning methods still cannot achieve low redundancy data and workload-aware hybrid data partitioning, mainly due to the following three limitations.

First, the state coding of the DRL-based horizontal partition study only considered the partition key and failed to consider the number of horizontal partitions. HP-DRL [10] encodes the partition key as 1 and the others as 0. This results in the same encoding for two and three partitions with a_2 as the partition key (shown in Fig. 2(a)). Similarly, VP-DRL [8] encodes the column groups as 1 and the others as 0. Although the partition results of connecting different attributes as column groups are different, the encoding vectors are the same (shown in Fig. 2(b)). Therefore, accurate partition state representation is the foundation of using DRL to implement a hybrid partition.



(a) HP-DRL [10] (b) VP-DRL [8]

Fig. 2. Different partition states with the same vector.

Second, the workloads of real applications are never static. It calls for automatically updating partitioning policies in response to workload changes. Thus, workload prediction is necessary to adjust strategy. However, the existing DRL method cannot adjust the strategy in time to cope with changes in workload. To be fully autonomous, the hybrid partition advisor must be able to predict workload in future.

Third, training the hybrid partition advisor for complex workloads requires a large amount of training data. Moreover, when monitoring workloads, it's necessary to analyze the potential performance of recommending a new partitioning strategy to determine whether it's worthwhile, as repartitioning can incur significant performance overhead. However, existing methods need to actually partition the database to train agents and evaluate the partition performance, which will waste a lot of time and resources. It calls for analysis method that can analyse the merits of partitioning strategies without actually partitioning in databases.

In summary, only by solving these three problems of partition state representation, workload prediction, and partitioning strategy performance analysis can we take advantage of DQN to solve the hybrid partitioning problem for dynamic OLAP workloads. But solving all three problems completely is still challenging.

Challenges. **(C1)** How to model the hybrid partition problem and represent the partition state? **(C2)** How to predict workloads accurately and adjust partitioning strategies in advance? **(C3)** How to analyse the performance of hybrid partitioning strategies without actually partitioning in databases?

In this paper, we present WorthyPar, a workload-aware data hybrid Partitioning advisor with deep reinforcement learning. To the best of our knowledge, this is

the first attempt to achieve self-driving hybrid partitioning relying on DRL for OLAP in a database. To make our method as efficient as possible, we: i) simplify the complex state space and define *Partition Line* to represent hybrid partition state accurately and trains an agent to recommend the partitioning strategy **(for C1)**, ii) train an ensemble learning model to predict workloads for various prediction horizons accurately **(for C2)**, and iii) train a lightweight model to analyze partition solutions performance based on three types of features to reduce the actual database partition **(for C3)**. Through these efforts, WorthyPar achieves the automatic derivation of suitable hybrid data partitioning strategies under dynamic workloads.

Contributions. The main contributions of this paper are the following:

(1) We first prove the *hybrid partition* problem is an NP-hard problem, model it to a DRL problem and give the representation of the partition state to recommend fine-grained partitioning strategies.

(2) We propose an ensemble-based method to predict query workload and adjust partitions automatically for dynamic workloads.

(3) We present a lightweight analysis model to analyze hybrid partition strategies without actually partitioning in databases.

(4) We use standard workload benchmarks to evaluate our advisors. Extensive experiments show that our advisor is up to 1.8× faster than the state-of-the-art method for varying the number of queries.

2 Preliminary

Since the hybrid partitioning strategies can reduce access to redundant data, in the following, we first give the definition of the *partition line* and the *hybrid partitioning problem*, and then prove the *hybrid partition* problem is an NP-hard problem.

Definition 1 (Partition Line). *A partition line represents the candidate location for horizontally or vertically partition table. Consider a table T with tuples $\{t_1, t_2, ..., t_m\}$ and attributes $\{a_1, a_2, ..., a_n\}$, divide m tuples heuristically into b blocks, candidate horizontal partition line is denoted by $l_{Th} = \{l_{Th_i} | 0 < i \leq b-1\}$ and candidate vertical partition line is denoted by $l_{Tv} = \{l_{Tv_j} | 0 < j \leq n-1\}$. The set of partition lines for T is denoted by $\mathbb{L}_T = \{l_T | l_T \in l_{Th} \cup l_{Tv}\}$.*

For example, Fig. 1(a) shows the table T, candidate horizontal partition line is denoted by $l_{Th} = \{l_{Th_1}, l_{Th_2}, l_{Th_3}\}$ and candidate vertical partition line is denoted by $l_{Tv} = \{l_{Tv_1}, l_{Tv_2}, l_{Tv_3}, l_{Tv_4}\}$. We can get Fig. 1(d) when $\mathbb{L}_T = \{l_{Th_2}, l_{Tv_1}, l_{Tv_4}\}$.

Definition 2 (Hybrid Partition Problem). *Given tables $\{T_1, T_2, ..., T_N\}$ with partition lines $\{\mathbb{L}_{T_1}, \mathbb{L}_{T_2}, ..., \mathbb{L}_{T_N}\}$, workload$\{q_1, q_2, ..., q_M\}$, predefined threshold of workload performance W and maximum number of partition lines C. $\mathbb{L}_{T_N}$ denotes all the partition lines in the table T_N, in which 1 indicates the partition line will be used, while 0 means no such partition line. The goal of Hybrid Partition Problem is to find $\mathbb{L}_{T_a}$ for each table T_a to vertical and horizontal partition*

them, such that the performance of the given workload (e.g. Runtime) using $\mathbb{L}_{T_a}$ not less than W and the size of $\mathbb{L}_{T_a}$ is not more than C.

Theorem 1. *The hybrid partitioning problem is NP-hard.*

Proof. The problem is clearly in NP, since the performance of the given workload (e.g. Runtime) using $\mathbb{L}_{T_a}$ could be checked in polynomial time.

To show the NP-completeness, we can build a reduction from the Knapsack problem in polynomial time. The Knapsack problem is one of the Karp's 21 NP-complete problems [19]. The definition of 0-1 Knapsack problem is as follows [1].

There are n kinds of items denoted by M_1, M_2, ..., M_n, and let $M = \{M_1, M_2, ..., M_n\}$. Each kind of item j has a weight S_j and a value V_j. It is assumed that all weights and values are nonnegative. The maximum weight can be carried in the bag is K, and E is defined as the predefined threshold of the total value of items, where K and E are both nonnegative values. The number X_j of copies of each kind of item is restricted to zero or one. The object of 0-1 Knapsack problem is to find a subset of items (equivalent to assigning value to each X_j), subjecting to the following two conditions:

$$\sum_{j=1}^{n} S_j X_j \leq K$$

$$\sum_{j=1}^{n} V_j X_j \geq E, X_j \in \{0, 1\}$$

We build a reduction from the Knapsack problem including three steps as follows.

Step 1: Instance Construction. We first construct a specific set $\mathbb{L}_{T_a} = \{L_{T_1}, L_{T_2}, ..., L_{T_n}\}$ in hybrid partition problem. The set $\mathbb{L}_{T_a}$ is consist of n partition lines. The number of each partition line j is S_j. The performance of the given workload using partition line j is V_j. Let predefined threshold C be equal to K, and let the predefined threshold W be equal to E. **Step 2: If the solution of 0-1 Knapsack problem exists, then the solution of the hybrid partition problem also exists.** If the solution of 0-1 Knapsack problem exists, i.e., $\exists M' \subseteq M$, such that

$$\sum_{M_j \in M'} S_j \leq K$$

$$\sum_{M_j \in M'} V_j \geq E, 1 \leq j \leq n$$

Then $\exists \mathbb{L}'_{T_a} \subseteq \mathbb{L}_{T_a}$, such that

$$\sum_{L_{T_j} \in \mathbb{L}'_{T_a}} S_j \leq C$$

$$\sum_{L_{T_j} \in \mathbb{L}'_{T_a}} V_j \geq W, 1 \leq j \leq n$$

Therefore, there exists a specific set $\mathbb{L}_{T_a}$ for each table T_a so that the performance of the given workload (e.g. Runtime) using $\mathbb{L}_{T_a}$ not less than W and the size of $\mathbb{L}_{T_a}$ is not more than C. Consequently, the solution of the hybrid partition problem instance exists. **Step 3: If the solution of the hybrid**

partition problem exists, then the solution of 0-1 Knapsack problem also exists. If the solution of the hybrid partition problem exists, i.e., there exists a specific set $\mathbb{L}_{T_a}$ for each table T_a so that the performance of the given workload (e.g. Runtime) using $\mathbb{L}_{T_a}$ not less than W and the size of $\mathbb{L}_{T_a}$ is not more than C. Let PL denote the set of partition lines involved in the $\mathbb{L}_{T_a}$. Let the function $F(PL)$ denote the total performance of the given workload only using partition lines in PL, and let N_{PL} denote the number of partition lines in PL. Then we can get that $N_{PL} \leq C$, $F(PL) \geq W$. We divide into two subsets: PL_1 and PL_2, subject to $PL = PL_1 \cup PL_2$, $PL_1 \cap PL_2 = \emptyset$. PL_1 and PL_2 are defined by $PL_1 = \bigcup L_{T_j}, L_{T_j} \subseteq PL, 1 \leq j \leq n$, $PL_2 = PL - PL_1$. As PL_2 cannot be used to partition any table, we can see that $f(PL_2) = 0$. So $f(PL) = f(PL_1) + f(PL_2) = f(PL_1) \geq W$. On the other hand, as PL_1 is the subset of PL, $N_{PL_1} \leq N_{PL} \leq C$. Therefore we can find the subset of items M', such that

$$\sum_{M_j \in M'} S_j = N_{PL_1} \leq K$$

$$\sum_{M_j \in M'} V_j = F(PL_1) \geq E$$

Consequently, the solution of 0-1 Knapsack problem exists.

Based on the three-step proof above, we conclude that the hybrid partition problem is an NP-hard problem.

3 Methodology

3.1 Overview

In order to design subtle partitioning strategies for dynamic workloads, we propose WorthyPar, a workload-aware hybrid partitioning advisor. We give the overview of WorthyPar in Fig. 3 and describe the high-level design of three main modules, i.e., *Hybrid Partition Module, Workload Monitoring Module* and *Performance Analysis Module*.

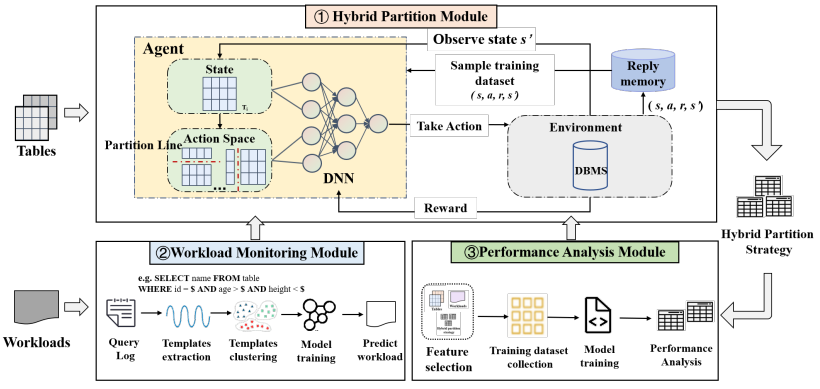


Fig. 3. The WorthyPar Overview.

Partitioning Workflow. When the data tables and query workloads are given, WorthyPar gets the initial state encoding and the action space from them. It trains the DRL agent and gets the recommended partitioning strategy, which can be analyzed for its effectiveness using the performance analysis model. Meanwhile, the query workload is counted in the query log. It can predict future workload changes through workload prediction module. Based on the predicted workload, the agent can be adjusted to recommend a new partitioning strategy. Then, in order to avoid actual partitioning of the database, the adjusted partitioning strategy is also analyzed using the performance analysis model. Thus, based on the analysis results, we can determine whether to actually adjust the partitioning or not.

Hybrid Partition Module. The module first uses frequent item set mining technology to determine the sort key, and uses tuple blocks as the granularity for selecting partition lines. The core idea of this module is to train a partitioning advisor using DRL techniques. It can obtain an initial state encoding and action space when given data tables and workload. Next, it trains the DNN to obtain an approximate Q-function based on the reward function and experience replay memory. Finally it can recommend a partitioning strategy with better performance.

Workload Monitoring Module. The workload forecasting module uses historical query logs for workload prediction. First, it extracts query templates from the logs. Then, it clusters the query templates and builds a query reach rate ensemble prediction model to predict future workloads. Based on the predict result, it can adjust the partitioning strategies.

Performance Analysis Module. The performance analysis module is mainly designed to analyze the performance of partitioning strategy. Firstly, we select the features that affect the database performance such as workload, data table, partitioning strategy. Then, we perform feature engineering and obtain the labeled data such as query scan size. Next, we train the lightweight performance analysis model. With the trained model, we can get the cost of partition policy to calculate reward. We can predict the performance of a partitioning strategy without actual partitioning and decide whether to adopt the strategy or not.

3.2 Hybrid Partition

We have proved in Section 2 that the hybrid data partition problem is NP-hard, since the traditional heuristics such as greedy methods cannot be adapted to the cost observed during search, there should be a trade-off between exploitation and exploration when choosing a partitioning strategy. We model the process of data partition as a MDP and solve the problem with DRL efficiently. In the following, we first discuss the DRL model for the hybrid partition problem, including states representation, actions, and rewards. Then we describe how to efficiently train agents.

Hybrid Partition as DRL. To formulate the hybrid partition problem as a DRL problem, we model tables as states and all possible partition operations on the table as actions. The reward is the performance gain for given workload.

States. Since we focus on hybrid partition strategies, it is important to represent the partition of tables. As described in the previous section, hybrid partition mainly consists of horizontal partition for certain tuples and vertical partition for certain attributes. Therefore, we use the one-hot encoding $s(T_i) = (l_{th1}, l_{th2}, ..., l_{th(b-1)}, l_{tv1}, l_{tv2}, ..., l_{tv(n-1)})$ to represent the partition of Table T_i. The $l_{th(b-1)}$ and $l_{tv(n-1)}$ encoded 0 or 1 to represent whether a partition line is used for horizontal or vertical partition. Q-learning is a simple off-policy algorithm for temporal difference learning, corresponding to a model-free Reinforcement Learning method [12] used when the agent initially only knows the set of possible states and actions.

Challenges: Specifically, in scenarios with massive amounts of data, a table may contain millions of tuples. There are two challenges with horizontal partition line encoding. (1) How to control the number of encodings to ensure scalability? (2) How to prevent the invalidation of the mapping from horizontal partition lines to tuples when tuples are deleted? To address the mentioned challenges, we have deployed *tuple blocks* as the granularity for horizontal partitioning.

Key idea: Since OLAP typically handles large scale and complex range queries, WorthyPar first used frequent itemset mining techniques to determine the sorting keys for automatically sorting the data. Then the sorted tuples are divided into 10 *tuple blocks*. In this way, the encoding representing the horizontal partition lines requires only 9 bits, addressing the scalability limitations caused by tuple level encoding. Additionally, it also can avoid the invalidation of the mapping from horizontal partition lines to tuples when tuples are deleted.

Action. we support two types of actions: (1)horizontal partition by the partition line when $\mathbb{L}_T = \{l_T | l_T \in l_{Th}\}$. (2) vertical partition by the partition line when $\mathbb{L}_T = \{l_T | l_T \in l_{Tv}\}$. Similar to state encoding, we still use one-hot encoding actions. For instance, the table T is hybrid partitioned in Fig. 4, the initial state

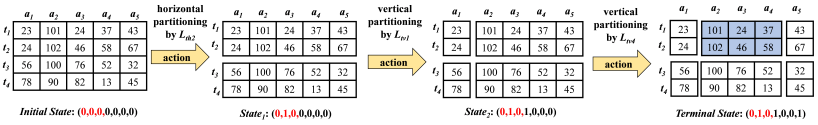


Fig. 4. An example of state representation.

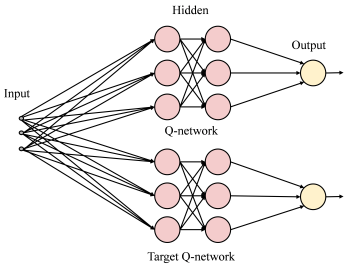
vector is $(l_{th1}, l_{th2}, l_{th3}, l_{tv1}, l_{tv2}, l_{tv3}, l_{tv4}) = (0, 0, 0, 0, 0, 0, 0)$. When it is horizontally partitioned by l_{th2}, its state vector is $(l_{th1}, l_{th2}, l_{th3}, l_{tv1}, l_{tv2}, l_{tv3}, l_{tv4}) = (0, 1, 0, 0, 0, 0, 0)$. When it is vertically partitioned by l_{tv1}, its state vector is $(l_{th1}, l_{th2}, l_{th3}, l_{tv1}, l_{tv2}, l_{tv3}, l_{tv4}) = (0, 1, 0, 1, 0, 0, 0)$. Finally, it is vertically partitioned by l_{tv4}, and its terminal state vector is $(l_{th1}, l_{th2}, l_{th3}, l_{tv1}, l_{tv2}, l_{tv3}, l_{tv4}) = (0, 1, 0, 1, 0, 0, 1)$.

The state s and an action a are then used as input to the neural network to predict the Q-value $Q(s, a)$.

Reward. The overall goal of the hybrid partition advisor is to find a suitable partition strategy for the workload that minimizes the sum of the partition block sizes accessed by the query. Therefore, we set the reward function to learn a DQN agent as follows.

$$r = -1 + last\_cost/cost \tag{1}$$

where *last_cost* is the cost of the last partition policy accessed by all queries in the workload, and *cost* is the cost of the partition policy accessed by all queries in the workload. They are all derived through the performance analysis module.



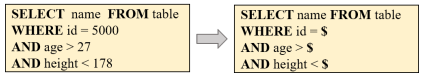


Fig. 6. An Example Query Template Extraction.

Fig. 5. Neural networks.

Training Procedure. Traditional Q-learning performs poorly when the state and action space is large, since searching for the appropriate state in a large table is a time-consuming task. DQN can solve the above problems well.

We constructed two same neural networks. The architecture of each network is shown in Fig. 5. In order to improve the non-linear mapping ability of the network, we added two hidden layers. During network training, the DQN chooses whether to partition the data horizontally or vertically in the current state. We set each training episode to a series of steps. We propose Hybrid Partitioning Training Algorithm.

Algorithm 1: Hybrid Partitioning Training

Input: Table T, Wokload W, Replay memory D, Batch size bs, Random probability ϵ, Learning rate α, Depreciation rate γ
Output: Trained Model parameters θ'

Initialize Q-network Q with random weights θ;
Initialize target Q-network Q' with random weights θ';
for *each episode* **do**
  initialize state
for *each step in episodes* **do**
  With probability ϵ select $a_t = argmax_a Q_\theta(s_{t+1}, a)$, otherwise random action a_t ;
  Execute action a_t ;
  Compute reward with $r = -1 + last\_access\_num/access\_num$;
  Store transition (s_t, a_t, r_t, s_{t+1}) in D;
  Sample $bs(s_i, a_i, r_i, s_{i+1})$ from D;
  Train Q-network with SGD and loss $\sum_{i=1}^{b}(r_i + \gamma \arg\max_{a \in A} Q'_\theta(s_{i+1}, a_i) - Q_\theta(s_i, a_i))^2$;
  Respect to the weights θ ;
  Every H steps reset target Q-network $\theta' = \theta$;
return

As we can see in Algorithm 1, given table T, workload W, replay memory D, batch size bs, random probability ϵ, learning rate α, depreciation rate γ, at

the beginning, we initialize Q-network Q with random weights θ and the target Q-network Q' with random weights θ'. Training consists of many episodes in Lines 3-13. During the training process, θ is continuously adjusted to adapt to the environment to achieve better performance. Finally, we obtain the trained parameters θ'.

After training all episodes, the neural network is adjusted to the optimal state. And the models of different episodes are saved for easy loading and using again without retraining. The time complexity of Algorithm 1 is $O(M * N)$, where M is the total number of training episodes and N is the total number of steps in each episode.

3.3 Workload Monitoring

To address the challenge of adjusting partition strategy, we propose a model to predict the arrival rate of query workloads. To accurately predict flexible workloads, we process query logs and build an ensemble learning model to train. In the following, we first propose the extraction and clustering of template from historical query logs and then describe the training process of the workload prediction model.

Query Template Extraction. To reduce prediction overhead, we can combine these SQL with similar syntactic formatting into a single query template. We format these statements to standardize them and remove numerical values so that SQL with similar filters can be combined into a single query template. An example of query template extraction is shown in Fig. 6.

Algorithm 2: Query Template Clustering

Input: *templates* is the new templates set, threshold α
Output: $clusters = \{c_1, c_2, ..., c_m\}$ is the cluster set

```
1  templates marked as unprocessed ;
2  for i=0; i <templates.num ; i++ do
3      if template_i is the first query then
4          Create c_m for template_i;
5      else
6          Calculate the DTW between template_i and each center of clusters;
7          if DTW_i between template_i and center_m of clusters is minimum and
             DTW_i <α then
8              c_m ← template_i;
9          else
10             Create c_m for template_i;
11     if the DTW between template_i and center_m ≥ α then
12         Drop template_i from c_m ;
13         Repeat Line3-14 for the template_i ;
14     if C_m does not receive a template for a long time then
15         Drop c_m from clusters;
16 return
```

Query Template Clustering. To avoid misjudging similar templates as dissimilar due to misaligned time axes, we propose a query template clustering

algorithm based on Dynamic Time Warping (DTW) [20]. We introduce a threshold α to the similarity measure and only query templates with a similarity less than α will be assigned to a new cluster.

The query template clustering algorithm is shown in Algorithm 2. For each new template, the distance between its historical arrival rate and the center of any group is calculated first, and the template is assigned to the cluster with the smallest distance and less than α. To accelerate the clustering, we use the kd-tree to efficiently find the closest center of existing clusters to the template in a high-dimensional space [2]. If there are no existing clusters (this is the first query), or if no cluster has a center close enough to the template, it will create a new cluster with the template as the only member (Lines 1-10). If the similarity of a template is no longer less than α, it will be removed from the current cluster and repeat Lines 1-10 to find a new cluster(Lines 11-13). If the cluster does not receive a template for a long period of time, it will delete the cluster(Lines 14-15). The time complexity of these steps is $O(nlogn)$, where n is the number of templates in the workload.

Forecasting Models Training. To combine the advantages of DLinear and Transformer, we use an ensemble learning model with a combined DLinear and Transformer, which can significantly improve the query arrival rate prediction accuracy through experimental analysis. The Ensemble combines the prediction results of the DLinear and Transformer models through the weight coefficient. The weight coefficient is calculated by the minimum sum of squared errors, and the model with higher prediction accuracy receives greater weight. The weights are calculated as shown in Eq.(2).

$$W_i = E_i / \sum_{i=1}^{n} E_i \tag{2}$$

where W_i is the weight of $Model_i$. E_i is the sum of squared errors of $Model_i$. n represents the types of prediction model. Here, n includes Transformer and DLinear.

With the trained workload prediction model, we obtain the composition of workloads in the future. If the workloads are the same as the historical workloads, we can directly change to the corresponding partitioning strategy. If the workloads are new, we can use incremental learning to get the corresponding rewards of the new workloads and incrementally train the agent without retraining all of them.

3.4 Performance Analysis

Analyzing the performance of hybrid partitioning strategies serves two purposes: it is used to calculate the reward for training the hybrid partitioning model, and it is also used to determine whether it is worthwhile to adjust partitioning schemes based on workload monitoring results.

Feature Selection and Collection. Feature selection is the foundation for the analysis model of the partition strategy training. Many features may affect

query performance during data partition. It is challenging to find the features that are most relevant to query performance.

Workload features. (1) *workload size* - The number of queries in workload. (2) *query num* - The workload includes the count of different queries. (3) *parse time* - The parsing time of the workload in database. (4) *table num* - The number of tables involved in workload. (5) *select num* - The number of columns involved in the SELECT predicate of the workload. (6) *where num* - The number of columns involved in the WHERE clauses of the workload.

Data partitioning strategy features. (1) *block size* - The total sizes of partition blocks of query results. (2) *block num* - The total number of partition blocks of query results. (3) *partition num* - The total number of partition blocks.

Training dataset collection. After determining the features, we need to gather training data to train performance analysis model. Each example includes the query execution time (as labeled data), workload, and partitioning strategy.

Model Training. In order to fully utilize hybrid data partition to enhance the efficiency of OLAP execution, the performance analysis has to be as fast as possible, which requires the training process of the performance analysis model to be efficient while ensuring accuracy. Fast performance analysis supports in training partition agents and decide partition strategies efficiently. So, the lightweight regression model we chose is XGBoost learner in model training module [3], which is a trade-off between performance and prediction accuracy. The loss function adopts the common loss function of XGBoost model, and its general form is as follows:

$$Obj^{(t)} = \sum_{i=1}^{n} l(y_i, y_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + constant \tag{3}$$

The regular term consists mainly of two parts: T represents the number of leaf nodes, and ω represents the fraction of leaf nodes.

4 EXPERIMENTAL EVALUATION

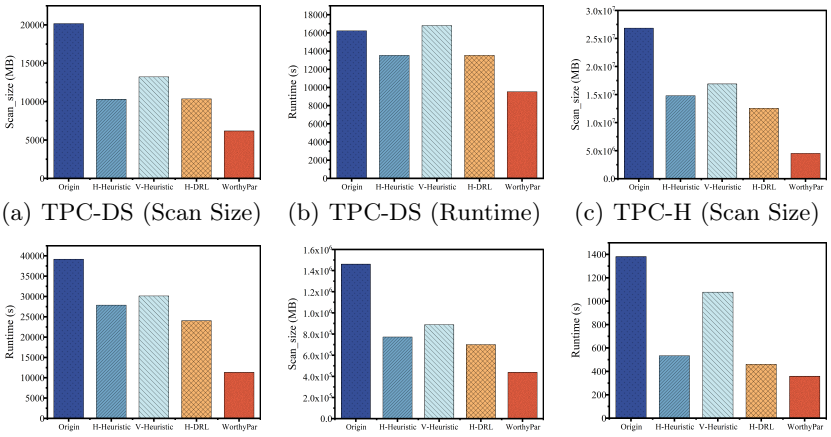
4.1 Experiment Setting

We used PostgreSQL 15.4 as the relational DBMS system, Python 3.7 and PyTorch 1.13.1 for developing and testing WorthyPar on a server with 4 RTX A6000 (48GB) GPUs, Intel(R) Xeon(R) Silver 4210R CPU, and 504GB RAM.

Data and Workloads: (1) **TPC-DS [21]** (2) **TPC-H [18]** (3) **JOB [16]** **Baseline Methods:** (1) ***Origin***. Original table without partition method means that no partition operation is performed on the table. (2) ***H-Heuristic***. Horizontal partition based on heuristic use the aprioristic algorithm to extract frequent item sets. (3) ***V-Heuristic***. Vertical partition based on heuristic method. (4) ***H-DRL***. This is a learned horizontal partition advisor based on DRL for OLAP-style workloads by Hilprecht et al [10]. **Evaluation Metrics:** (1) **Scan size** measures the average I/O cost. (2) **Run time** measures the cost of database process queries.

4.2 Evaluation on Hybrid Partition

To evaluate the hybrid partitioning method, we compared the scan size and runtime of WorthyPar against baselines on three datasets with workload of 1000 queries. The results are shown in Fig. 7. Since ***Origion*** does not perform partitioning, its scan size and runtime are always the highest. In contrast, the reinforcement learning based partitioning method demonstrates better performance than heuristic based methods. The WorthyPar outperforms ***H-DRL***. Because its ability to perform partitioning actions on both rows and columns.



(a) TPC-DS (Scan Size) (b) TPC-DS (Runtime) (c) TPC-H (Scan Size)

(d) TPC-H (Runtime) (e) JOB (Scan Size) (f) JOB (Runtime)

Fig. 7. WorthyPar vs. Baselines.

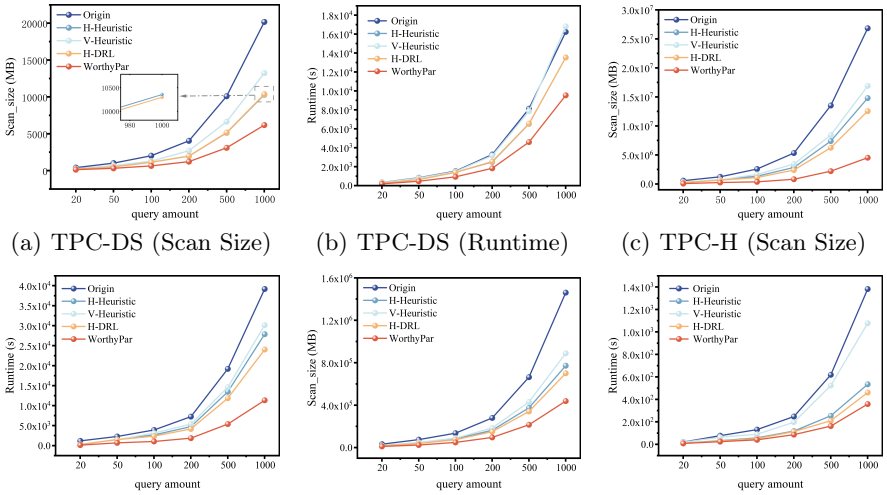
To further explore the heuristic methods and the optimization capabilities based on DRL, we benchmarked against the original method, comparing the optimization effects of the four methods. We can conclude that WorthyPar exhibits the best optimization performance in reducing workload run time across all datasets from Table 1. The partitioning performance of WorthyPar is superior to other methods.

Table 1. The runtime of WorthyPar on three datasets of varying workload sizes.

Datasets	Method	Workload RunTime(s)						Reduction
		20	50	100	200	500	1000	
TPC-DS	Origin	331.5	828.5	1527.8	3278.9	8126.0	16233.1	-
	H-Heuristic	256.9	628.3	1390.6	2496.8	6519.5	13534.2	19.3%
	V-Heuristic	318.1	794.1	1483.2	3168.8	7856.9	16810.3	2.3%
	H-DRL	257.5	631.6	1390.9	2536.3	6544.7	13534.6	18.9%
	WorthyPar	184.9	458.9	926.2	1822.7	4590.0	9533.1	**42.9%**
TPC-H	Origin	1174.3	2280.0	3903.2	7241.2	19195.8	39173.0	-
	H-Heuristic	300.2	1529.9	2674.1	4741.9	13461.0	27870.3	38.6%
	V-Heuristic	332.7	1487.6	3120.1	5390.6	14604.8	30130.9	33.2%
	H-DRL	269.6	1485.2	2359.3	4154.4	11845.3	24029.6	45.1%
	WorthyPar	104.7	680.7	1026.3	1858.6	5383.1	11328.3	**75.3%**
JOB	Origin	20.2	76.9	130.8	247.1	618.3	1381.2	-
	H-Heuristic	10.8	32.6	57.6	118.0	254.0	533.7	55.3%
	V-Heuristic	18.9	63.4	89.7	197.0	522.7	1076.9	18.8%
	H-DRL	10.1	30.1	50.8	113.2	210.0	460.7	59.8%
	WorthyPar	7.6	23.5	39.8	85.9	162.3	358.1	**69.1%**

4.3 Evaluation on Scalability

To validate the scalability of WorthyPar, we compared its performance with baselines under varying data and workload updates. Fig. 8 shows that as the number of queries in the workload increases, WorthyPar consistently outperforms the baselines. This is due to WorthyPar which is workload-aware partitioning method and the hybrid partition can to reduce redundant data access.

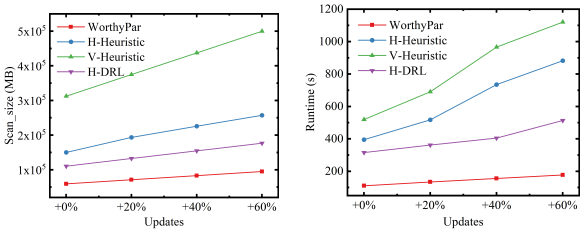


(a) TPC-DS (Scan Size) (b) TPC-DS (Runtime) (c) TPC-H (Scan Size)

(d) TPC-H (Runtime) (e) JOB (Scan Size) (f) JOB (Runtime)

Fig. 8. Scalability on Workload.

In Fig. 9, we maintain a workload of 20 queries and vary the size of the TPC-H dataset. We observe that WorthyPar consistently performs best even at a relatively high update rate of up to 60%. This is due to the scalability of our action and state encoding scheme used during the training of the DRL model.



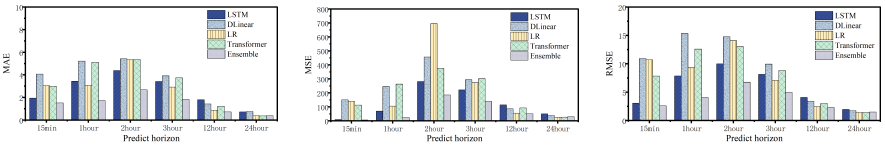
(a) Scan size on TPC-H (b) Runtime on TPC-H

Fig. 9. Scalability on Data.

4.4 Evaluation on Workload Monitoring

To evaluate the workload monitor, we conducted experiments on its prediction accuracy using three performance metrics: Mean Absolute Error (MAE), Mean Square Error (MSE), and Root Mean Square Error (RMSE). The results are

presented in Fig. 10. Although LSTM and Transformer models demonstrated competitive performance, our Ensemble model consistently outperformed these models, achieving the best prediction performance across the first five forecast horizons in all evaluation metrics. The results clearly indicate the potential of the proposed workload monitor to accurately predict workloads.

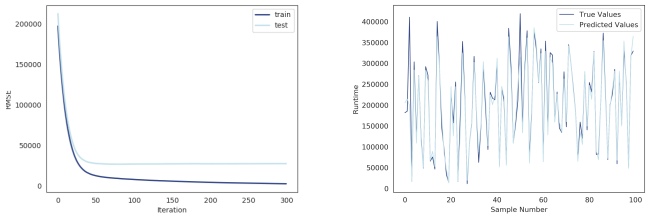


(a) MAE (b) MSE (c) RMSE

Fig. 10. Forecasting Model Evaluation.

4.5 Evaluation on Performance Analysis

We generate 1000 samples to train the performance analysis model, 80% of which were used as training set and 20% as test set. Then we trained XGBoost Regressor model. The learning curves of the model are shown in Fig. 11(a), where the RMSE of both the train and test datasets decreases with the number of iterations. The model almost converges at about 40 iterations. It shows that the performance analysis model has a good fitting ability. Fig. 11(b) shows the R Square of the model is 0.96. Performance analysis can accurately predict partition performance and avoid the overhead of repartitioning through the database.



(a) Learning curve. (b) True vs. predicted values.

Fig. 11. Scalability on Data.

5 conclusion and future work

In this paper, we propose WorthyPar, which consists of three main modules. *Hybrid Partition* refers to training a DRL agent to recommend hybrid partition strategies for different data tables and workloads. *Workload Monitoring* prompts the advisor to adjust the partition strategy in advance by predicting future workloads, and *Performance Analysis* predicts the performance of the partition strategy while avoiding the actual partitioning of the database. In future work, we plan to apply this method to complex distributed database scenarios to improve database query performance.

Acknowledgments. This work is supported by National Natural Science Foundation of China (NSFC) (62232005, 62202126), Natural Science Foundation Project of Heilongjiang Province (No YQ2024F005).

References

1. A.Levitin: Introduction to the design analysis of algorithms. In: Addison-Wesley (2003)
2. Bentley., J.L.: Multidimensional binary search trees used for associative searching. In: Communications of the ACM. pp. 509–517 (1975)
3. Chen, T., Guestrin, C.: Xgboost: A scalable tree boosting system. In: SIGKDD (2016)
4. Cui, S., Wang, H., Liu, X., Ding, X.: Tempsched: A temperature-aware storage scheduler for time series across cloud-edge-device. In: ICDE (2025)
5. Cui, S., Wu, X., Wang, H., Wu, H.: Multimodal data modeling technology and its application for cloud-edge-devicecollaboration. Ruan Jian Xue Bao/Journal of Software **35**(3), 1154–1172(in Chinese) (2024)
6. Ding, X., Song, Y., Wang, H., Yang, D., Wang, C., Wang, J.: Clean4tsdb: A data cleaning tool for time series databases. Proc. VLDB Endow. **17**(12), 4377–4380 (2024)
7. Ding, X., Wang, H., Li, G., Li, H., Li, Y., Liu, Y.: Iot data cleaning techniques: A survey. Intell. Converged Networks **3**(4), 325–339 (2022)
8. Durand, C., Gabriel, Piriyev: Automated vertical partitioning with deep reinforcement learning. In: Communications in Computer and Information Science. pp. 126–134 (2019)
9. H, L., PJ, L., TY, W.: Survey of intelligent partition and layout technology in database system. Ruan Jian Xue Bao/Journal of Software **33**(10), 3819–3843(in Chinese) (2022)
10. Hilprecht, B., Binnig, C., Rohm, U.: Towards learning a partitioning advisor with deep reinforcement learning. In: SIGMOD (2019)
11. Huang, K., Tao, Z., Wang, C., Guo, T., Yang, C., Gui, W.: Cloud-edge collaborative method for industrial process monitoring based on error-triggered dictionary learning. IEEE Trans. Ind. Informatics **18**(12), 8957–8966 (2022)
12. Irodova, M., Sloan, R.H.: Reinforcement learning and function approximation. In: In FLAIRS Conference. 455–460. (2005)
13. J, A., A, P., P, M.: Bridging the archipelago between row-stores and column-stores for hybrid workloads. In: Proc. of the 2016 Int'l Conf. on Management of Data (2016)
14. Kang, D., Jiang, R., Blanas, S.: Jigsaw: A data storage and query processing engine for irregular table partitioning. In: SIGMOD. pp. 898–911 (2021)
15. L, S., MJ, F.: Skipping-oriented partitioning for columnar layouts. In: VLDB (2016)
16. Leis, V., Gubichev, A., Mirchev, A., Boncz, P., Kemper, A., Neumann, T.: How good are query optimizers, really? (2015)
17. Li, Z., Ding, X., Wang, H.: An effective constraint-based anomaly detection approach on multivariate time series. In: APWeb-WAIM. pp. 61–69 (2020)
18. Poess, M., Floyd, C.: New tpc benchmarks for decision support and web commerce. In: International Conference on Management of Data (2000)
19. R.M.Karp: Reducibility among combinatorial problems. In: in R.E.Miller and J.W. Thatcher (Eds.), Complexity of Computer Computations. (1972)
20. T, R., B, C., Mueen A, e.a.: Searching and mining trillions of time series subsequences under dynamic time warping. In: SIGKDD explorations (2012)
21. TPC-DS benchmark: http://www.tpc.org/tpcds/
22. Wang, C., Qiao, J., Huang, X., Song, S., Hou, H., Jiang, T., Rui, L., Wang, J., Sun, J.: Apache iotdb: A time series database for iot applications. Proc. ACM Manag. Data **1**(2), 195:1–195:27 (2023)